目錄

从Python到Django入门教程



by boris_cn@263.net

废话少说，开始吧！

从Python到Django入门教程

前言

Python是一门面向对象的编程语言，它相对于其他语言，更加易学、易读，非常适合快速开发。Python具有简单、易学、免费、开源、可移植、可扩展、可嵌入、面向对象等优点，它的面向对象甚至比java和C#.net更彻底。作为一种通用语言，Python几乎可以用在任何领域和场合，角色几乎是无限的。Python在软件质量控制、提升开发效率、可移植性、组件集成、丰富库支持等各个方面均处于先进地位。学习编程语言，除了拿好现有的饭碗，还要选择学习业内目前最先进、最热门、将来应用最广泛、最有前途和前景的编程语言。有人预言，Python会成为继C++和Java之后的第三个主流编程语言。

Django是一个开放源代码的Web应用框架，由Python写成。采用了MVC的软件设计模式，即模型M，视图V和控制器C。它最初是被开发来用于管理劳伦斯出版集团旗下的一些以新闻内容为主的网站的。并于2005年7月在BSD许可证下发布。这套框架是以比利时的吉普赛爵士吉他手Django Reinhardt来命名的。

Django的主要目标是使得开发复杂的、数据库驱动的网站变得简单。Django注重组件的重用性和“可插拔性”，敏捷开发和DRY法则（Don't Repeat Yourself）。在Django中Python被普遍使用，甚至包括配置文件和数据模型。

在Python各种web框架中，Django的文档最完善、市场占有率最高、招聘职位最多！

本书整理Python和Django的入门教程，给零起点的程序员一个完整的学习路径：

```
Python 3.6--->Django 1.11
```

开发环境

本文的范例基于Visual Studio Code编辑器完成的。因此，你需要准备以下这些软件：

- VS Code编辑器，还需要安装Python扩展。
- Python3的安装文件，根据你自己的操作系统安装，"下一步&下一步"就能搞定，无需多说。
- 用virtualenv搭建Django的开发环境，后文会详细描述。

开始学习

根据自己的情况选择从哪里开始学习，如果你已经有了一定的基础，你可以选择跳过某些章节，对于大多数初学者来讲，学习的路径如下：

1. 搭建Python开发环境

2. 简明Python教程

3. Django step by step

本书托管在GitHub， 如果有问题请在线提交。

搭建Python开发环境

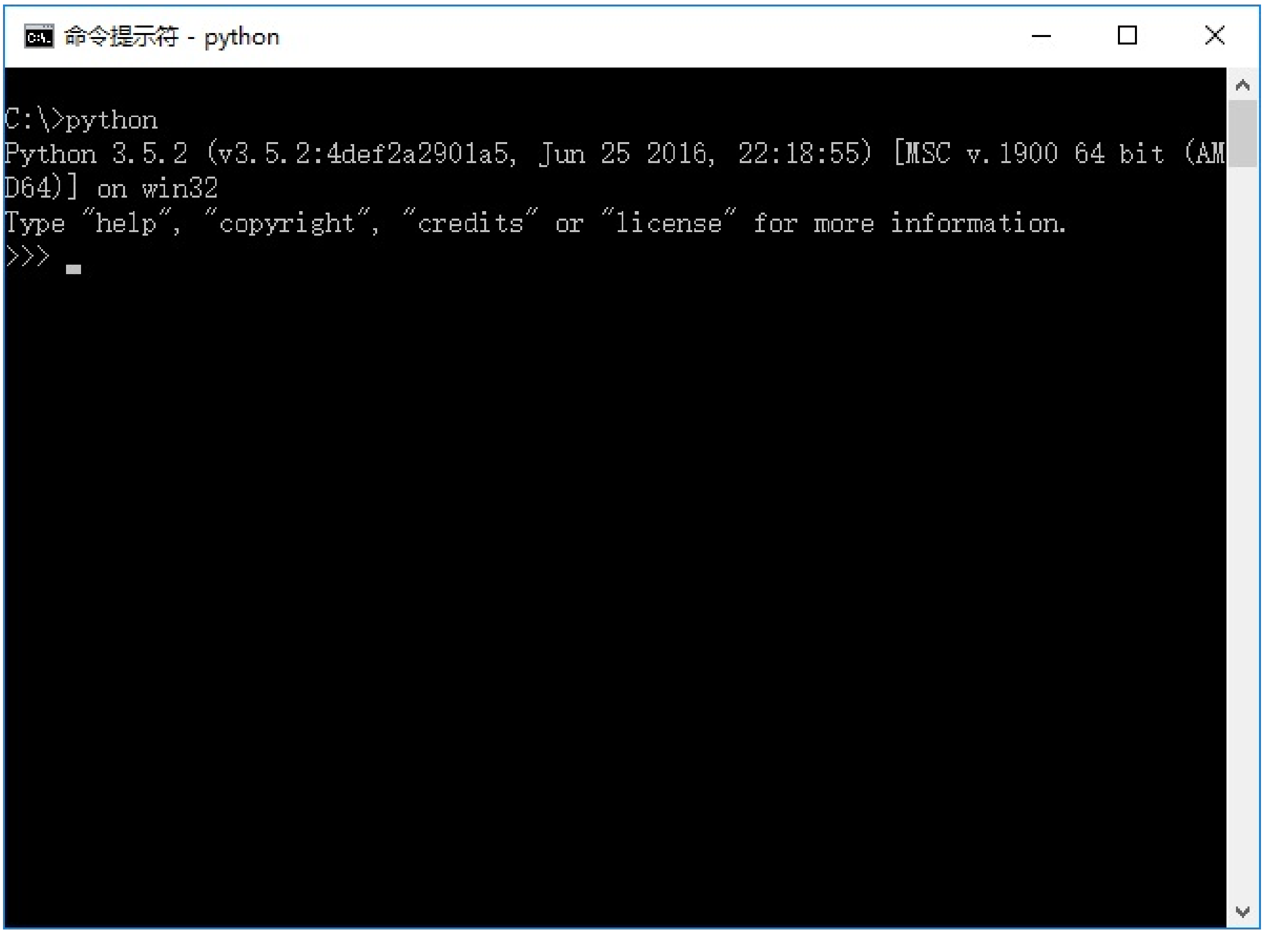
安装Python

Python的开发环境是比较简单的，到https://www.python.org/downloads/去下载对应的Python3安装包，安装的时候勾选上修改PATH的选项，然后完成安装就是了。本文使用的是Python 3.6.3。

调出你的终端（cmd），输入

```
python
```

如果出现下面的提示，那么说明你已经安装成功了。



Python环境搭好之后首先要做的就是这个：

```
print('hello world!')
```

恭喜你！

安装virtualenv

virtualenv可以建立多个独立的虚拟环境，各个环境中拥有自己的python解释器和各自的package包，互不影响。 使用Python自带的pip工具可以很方便的安装、卸载和管理Python的包。

```
pip install virtualenv
```

pip和virtualenv可以很好的协同工作，同时使用这两个工具非常方便。 用virtualenv env1就可以创建一个名为env1的虚拟环境了，进入这个虚拟环境后，再使用pip install安装其它的package就只会安装到这个虚拟环境里，不会影响其它虚拟环境或系统环境。 接下来我们要用这个工具创建我们自己的开发环境。

安装Django1.11

为了能够使用Django的命令行，我们把Django安装到系统的环境中，在命令行中输入：

```
pip instal django
```

就可以完成安装。

创建第一个Django项目

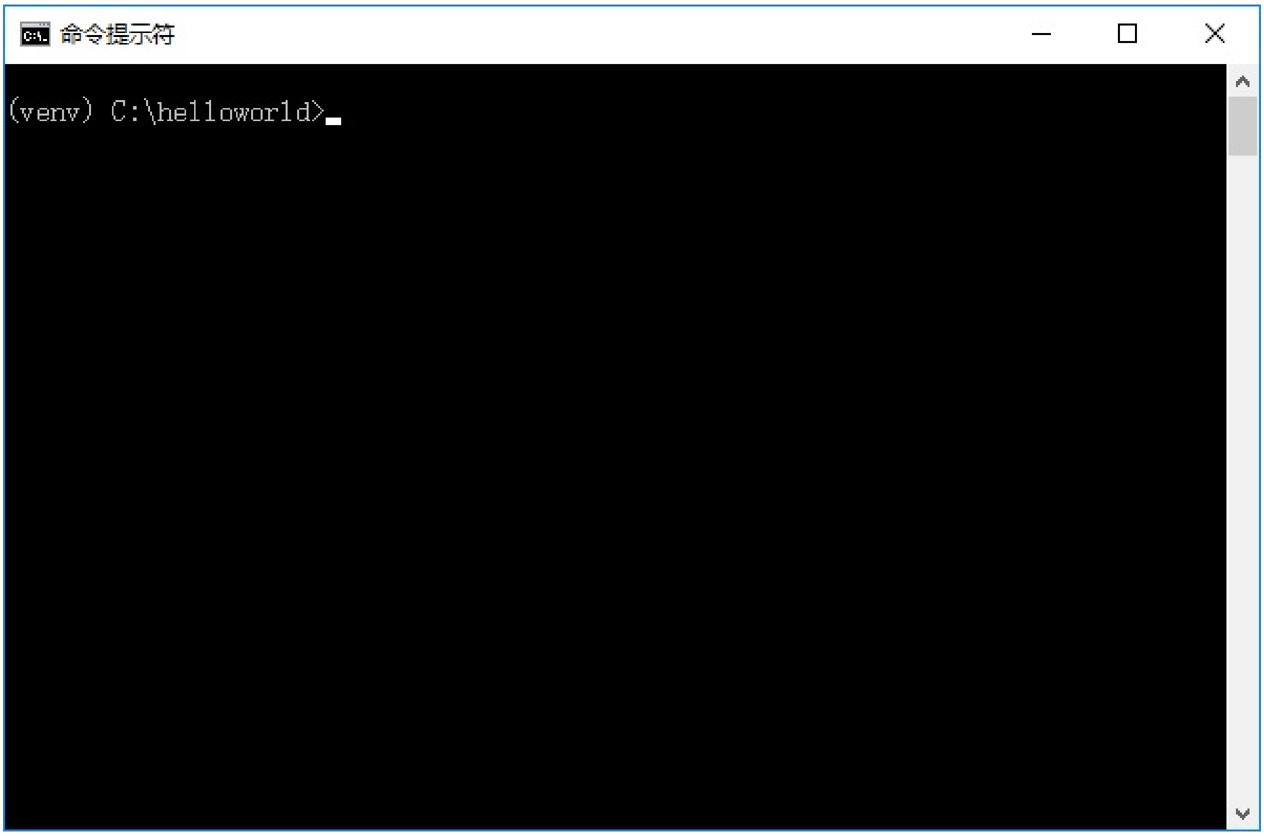
为了能够统一的管理工程的代码等信息，我们将工程代码和virtualenv环境都放在同一个目录中，这样无论这个目录拷贝到哪里，都可以直接加载环境之后开始运行，首先输入下面的命令创建第一个django项目：

```
C:\>django-admin startproject helloworld
C:\>cd helloworld
```

然后我们在helloworld目录下面创建一个venv的目录，保存这个Django项目的虚拟环境，之后我们在这个虚拟环境中安装Django1.10

```
C:\helloworld\>virtualenv venv
C:\helloworld\>venv\scripts\activate
(venv) C:\helloworld\>pip install django
(venv) C:\helloworld\>
```

这个时候你应该可以看到提示符前面增加了“(venv)”的字样，如下所示：



这个时候你的virtualenv就已经激活了，如果你再输入 `python` 命令： 的时候，就会使用这个虚拟环境下面的Python。请注意，下面的教程我么都是在这个环境下面运行的。

下面让我们运行一下我们的第一个Django项目。如果你运行了 `python` 命令，输入 `exit()` 退出。在命令行下面输入：

```
(venv) C:\helloworld\>python manage.py runserver
```

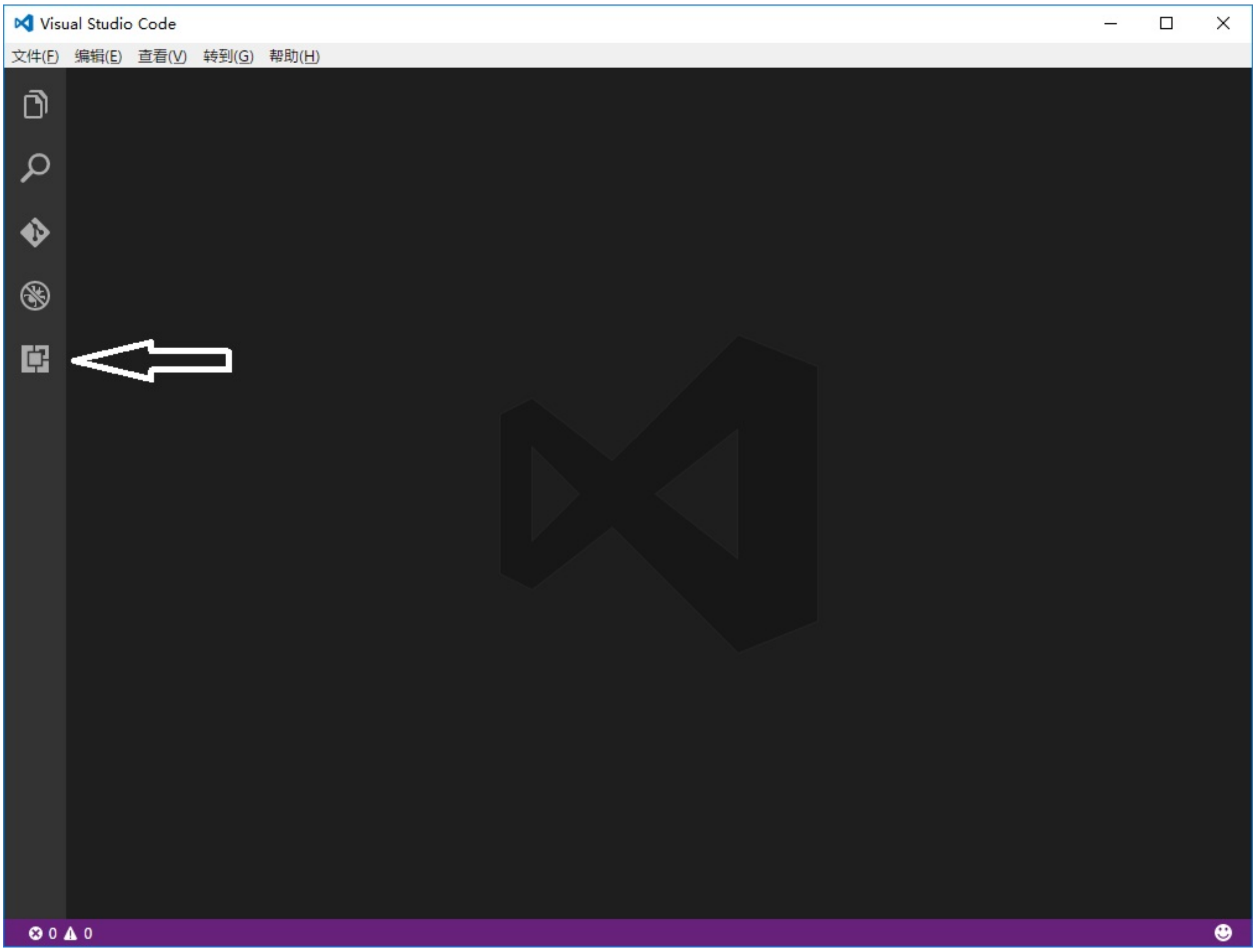
然后使用浏览器打开这个地址http://localhost:8000/就可以看到一个欢迎页面了。



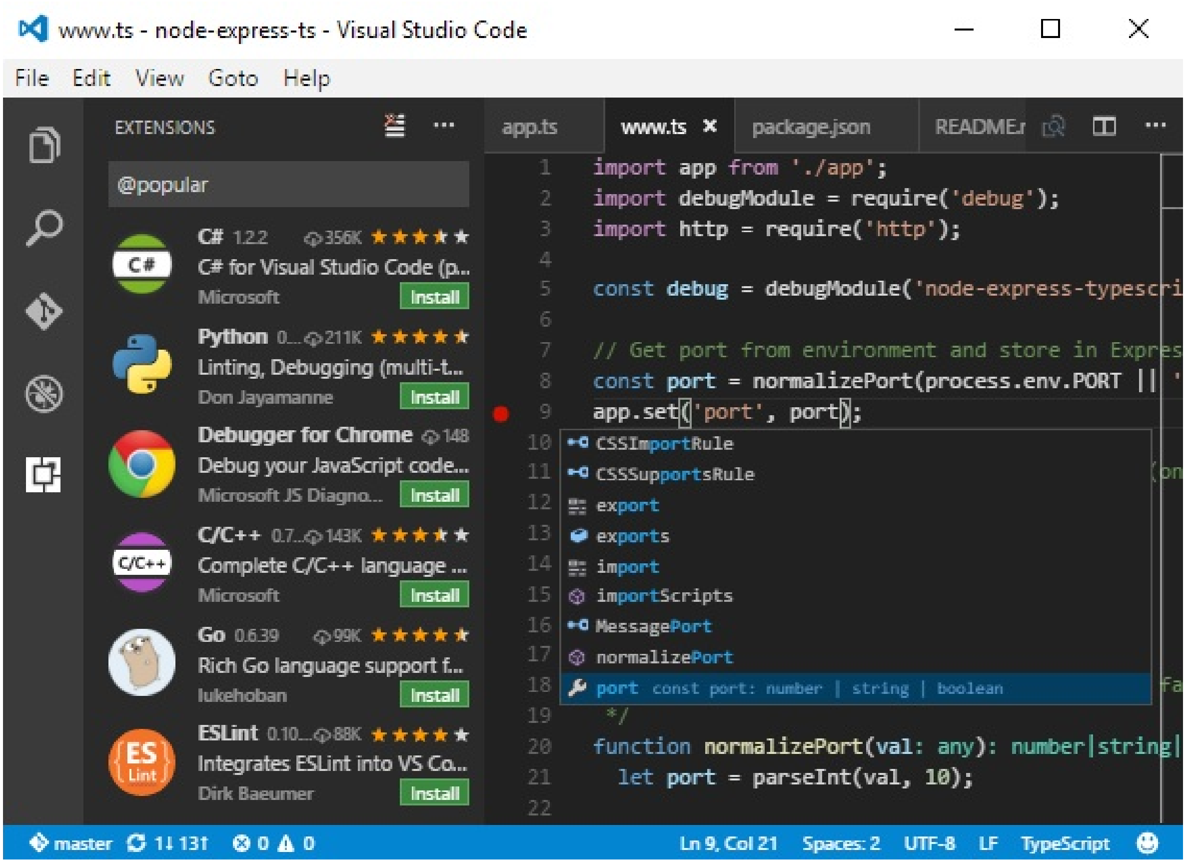
在命令行下面使用 `Ctrl+C` 可以退出这个Django项目。

安装Visual Studio Code作为Python IDE

访问https://code.visualstudio.com下载Visual Studio Code（简称VS Code）客户端，然后安装。打开后会看到如下的界面。



点击箭头所指的 `扩展` 按钮，输入 `@popular` ，就会显示最流行的扩展清单，选择Python扩展进行安装。



由于我们的virtualenv目录会放到Python项目的venv子目录下，所以我们要对Python扩展进行一点设置。在VS Code菜单中选择“文件”->“首选项”->“用户设置”，就会打开用户设置的文件 `settings.json` ，输入如下信息指定我们的Python环境路径：

```
{
    "python.pythonPath": "${workspaceRoot}/venv/scripts/python.exe",
}
```

使用VS Code打开helloworld

在helloworld目录下输入：

```
    (venv) C:\helloworld\>code .
```

即可使用VS Code打开helloworld工程。VS Code没有工程描述文件，一个目录就是一个工程，后面的例子我们都用这个开发工具完成。

完结！

继续阅读[简明**Python**教程]

简明Python教程

目 录

继续阅读简介

A Byte of Python

《A Byte of Python》是一本介绍用Python语言编写程序的免费书。它为Python初学者提供指导或指南。 如果你对计算机的知识仅限于知道如何保存文本文件，那么这本书非常适合你。

使用Python 3完成

本书介绍了如何使用Python 3进行编程。如果你仍然使用Python 2，本文经过改写后也可以作为Python 2的一本指南。

《A Byte of Python》的读者对象?

下面是一些人对本书的评价:

> 这是我见到的最好的初学者指南！感谢你的努力。 -- Walt Michalik

> 你做了我在网上发现的最好的Python指南，伟大的工作，谢谢！ -- Joshua Robin (mailto:joshrob@poczta.onet.pl)

> 为初学者做了Python编程的卓越介绍。 -- Shan Rajasekaran

> Python初学者最好的入门指南。 -- Nickson Kaigi

> 这本指南的每一页都让我愈发爱上这门编程语言。 -- Herbert Feutl

> Python初学者的一本完美的入门教程，为你打开Python的神秘之门。 -- Dilip

> 我正在全神贯注于我的工作，然后发现了这本"A Byte of Python"，一本非常棒的入门指南，有着非常棒的范例。 -- Biologist John

> 最近我开始阅读a Byte of python。非常棒的一部教程。我推荐给所有的Python程序员。 -- Mangesh

> 我正在阅读Swaroop写的《A Byte of Python》,我觉得对于初学者来说这是最好的入门指南，对于有经验的开发者也很有帮助。 -- Apostolos

> 很喜欢阅读Swaroop写的《A Byte Of Python》，这是我最喜欢的一本书了。 -- Yuvraj Sharma

> 感谢你写了这本《A Byte Of Python》。我刚开始通过它学习编程，大概两天后就可以写一些简单的小游戏了。你的这本指南非常棒，我想让你知道他很有价值。 -- Franklin

> 我来自印度Dayanandasagar工程学院。首先我想说"The byte of python"对于像我这样的初学者来说真是一本不错的入门指南。这本书讲述概念非常清晰，还附带有很多的小范例帮助我学习Python语言。太感谢了！ -- Madhura

> 我是一名18岁的在校学生，目前在爱尔兰的大学学习计算机。我非常感谢您写了这本 "A Byte of Python"，我已经学习过3门编程语言 - C, Java and Javascript, 由于你写的这本非常棒的入门教程，使得Python成为了我学习过的最简单的编程语言。这是我读过的最好的编程语言入门教材之一。感谢你，希望你继续这项伟大的工作。 -- Matt

> 嗨，我来自多米尼加共和国，我叫Pavel。最近，我读了你的《A Byte of Python》，我认为它是极好的!! :)。从例子中我学到了很多，你的书对像我这样的初学者有很大帮助... -- Pavel Simo

> 我来自中国，是一名在校生。我看完了你的这本《A byte of Python》，写的太好了！这本书如此之浅显易懂，非常适合于初学者。我之前对于Java和云计算非常感兴趣，经常在服务器端进行编程，现在我觉得Python对于我这样的情况来说非常适合。看完你的书之后，我觉得应该马上开始Python语言编程的实践。我的英语不太好，写这封邮件只想感谢你的辛勤工作。 -- Roy Lau

> 我读完了《A Byte of Python》，我想我真的应该感谢你，当我读到最后，我感到非常难过，因为我不得不再回到无趣的、乏味的学习笔记等中去。不论如何，作为Python学习手册，我欣赏你的书。 Samuel Young

> 亲爱的Swaroop，我正跟着一个对教学没兴趣的老师上课。我们使用的是 O'Reilly 的《Python学习手册（第二版）》，它不是没有任何编程知识的初学者学习的教材，而且我们的老师也应该以另外一种方式教学。非常感谢您的书籍，如果没有它，我就不能学会Python和编程。一百万次地感谢！您把知识"掰开揉碎"到初学者能够理解的水平，这不是所有人都能做到的。 -- Joseph Duarte

> 我喜欢你的书！这是最好的Python教程，非常有用的参考手册，才华横溢，真正的杰作！请继续这种好的工作！ -- Chris-André Sommerseth

> 首先，我要感谢你写了这本书。我想对于那些想要Python初学者指南的人来说这本书再适合不过了。 我大概2-3年前第一次听说这本书。当时我还没有办法阅读英文版本的书籍，所以我找到了一本中文翻译的版本，它带领我进入了Python编程的世界。 现在，我重新读这本书的英文版。我不敢相信我居然不用查字典就看完了所有的章节。当然了这归功于您使用非常浅显易懂的方式完成这本书。 -- myd7349

> 我给你发邮件，就是为了感谢你在线编写的《A Byte of Python 》。在偶然发现你的书之前，我已经尝试学习Python有几个月了。虽然我对pyGame取得了有限的成绩，但我从来没有完成过一个程序。 感谢你对分类的简单化，Python看起来确实是一个能够达到的目标。看起来我已经学会了基础知识，并能够继续我的真正目标——游戏开发。 ... 再一次，非常感谢你将如此结构良好而且有用的编程基础教程放到网上，它帮助我彻底理解了OOP，这之前两本书都没行。 -- Matt Gallivan

> 我感谢你以及你的《A Byte of Python》，我发现这是学习Python最好的途径。我现在15岁，住在埃及，我的名字叫Ahmed。Python是我学习的第二个编程语言，我在学校学习了Visual Basic 6，但我并不喜欢它，可是我真的喜欢学习Python。我成功地编写了地址簿程序。我打算尝试编写、阅读更多Python程序（如果你有意给我推荐一些有帮助的源代码）。我也将开始学习Java，如果你能告诉我在哪里可以找到向你的教程一样好的Java教程，那将对我有很大帮助。多谢。 -- Ahmed Mohammed

> 初学者想更多的学习Python编程的最好的资源是Swaroop C H编写的110页的PDF教程《A Byte of Python》。这本书写得很好，跟随它学习很容易，或许是当前可以得到的最好的Python入门教程。 -- Drew Ames

> 昨天，我在Nokia N800上浏览了《A Byte of Python》的大部分内容，这是我至今遇到的最简明扼要的Python教程,极力推荐作为学习Python的一个起点。 -- Jason Delport

> 对我而言，@swaroopch 编写的《A Byte of Vim》和《A Byte of Python》是最棒的技术作品，读起来非常棒。#FeelGoodFactor -- Surendran

> 《A Byte of python》至今最好 （在回答"谁能推荐一个又好又便宜的学习Python基础的资源？"中答道） -- Justin LoveTrue

> "《A Byte of Python》非常有用，多谢 :)" Chinmay

> 永远是对新手和有经验的程序员都适合的《A Byte of Python》的爱好者。 -- Patrick Harrington

> 我几天前开始通过你的书学习Python，这是一本非常好的书。写得如此之好，让我的生活非常美好。现在我已经是你的一个粉丝了，太感谢了！ -- Gadadhari Bheem

> 我在学习Python语言之前了解一些汇编、C、C++、C#和Java。我学习Python语言的原因是他非常流行而且强大。这本书对于有经验和无经验的程序员都是一本很好的教材，我花了10个半天看完了这本书，非常有启发！ -- Fang Biyi (PhD Candidate ECE, Michigan State University)

> 感谢作者写了这本书！ 这本书解答了我关于Python语言的很多问题，比如面向对象编程等。 看完之后我不觉得自己就成了面向对象的专家了，但是这本书领我入了门。 我现在已经写了很多的Python程序作为系统管理员的工具，他们都是面向过程的小程序。 最后再次感谢这本书的帮助。 -- Bob

> 我想感谢你写了这本书，我是从这本书开始学习编程的。Python现在是我的首选开发语言了。感谢你让我能够开发出很多有用的小工具，在学习Python编程之前这是我难以想象的。 -- "The Walrus"

> 我想谢谢你写了这本 *A Byte Of Python* 。这本书对于我学习Python语言编程来说是无价之宝。 我是一名刚刚入门的程序员，几个月前刚刚开始学习。之前我都是通过youtube的视频教程或者其他的在线免费教程学习。昨天我决定从你写的这本书开始入手学习。开始几页我就学习到了之前从来无法企及的编程知识。我有一个小问题，能不能给出一些最新的例子和解释，我非常期待继续更新。 感谢你不仅仅写了这本书，而且免费发布。感谢有你们这些无私的人能够帮助我们这些芸芸众生。 -- Chris

> 我在2011年的时候和你通信联系过，现在我重新又开始Python编程，我要再次感谢你写了这本"A Byte of Python"。如果没有他，我不知道该从哪里开始。我从那时起在我们公司里就用Python写了很多的程序。我不能说我是一个高级程序员，但是我还是可以帮助到周围的人。当我看到对于Byte的解释的时候，我就不再继续学习C/C++语言了，因为我发现开始就讲到了增量赋值这个概念没有讲清楚。当然也有一些对于增量赋值的描述，但是我发现很难理解。我并不是说C/C++语言很难学，或者我自己学不会，我只是说C/C++语言的文档并没有仔细的定义所出现的各种名词、概念，以至于学习起来非常费劲。就好像计算机无法理解非计算机术语一样，一个初学者刚开始接触一门知识首先需要了解基本的名词和概念，否则就会出现"蓝屏"。其实解决方法也很简单，找到这些名词和概念，给出合理的、简单的定义。感谢你写了这本非常的浅显易懂的书。我希望你能够继续完成全部的名词解释。Python的官方文档写的是不错，但是很难让初学者从一开始就读得懂。第三方的入门教程应该快速的解释一些单词的释义以便于后面继续讲解艰深的内容。我把你的书推荐给了身边的人，包括澳洲的、加勒比的、美国的。祝愿你在以后取得更大的成功1 -- Nick

> 我是ankush，今年19岁。我刚开始学习Python遇到了很大的困难，我翻了很多的书，但是都无法解决我的问题。直到我发现了你写的这本书，它让我爱上了Python，谢谢你！ -- Ankush

> 感谢你写了这么一本如此棒的Python入门教材。我是一名分子生物学的研究人员，编程的知识了解不多，但是我的工作需要处理大量的DNA测序的数据以及分析微观世界的图像。对于这两件事情来说，用Python编程非常有用，没有他我就无法完成这项为期6年的项目。 这么棒的教材居然免费提供，这说明世界上还是有好心人的啊！ :) -- Luca

> 如果这是你第一次学习编程语言，你应该看《A Byte of Python》。它确实从适当的角度介绍了Python程序设计，并且后续的深入教程也不是很难。最重要的是要动手写程序！ -- "{Unregistered}"

> 感谢你出版的这两本书：《A Byte of Python》和《A Byte of Vim》。在四五年前我刚开始学习编程的时候非常有用。现在我正在开发一个项目，这是我梦想中要实现的项目，我想说一声："谢谢你！"，希望你继续走下去，你是我前进的动力！ -- Jocimar

> 我三天前看完了《A byte of Python》，写的太好了！没有一个字的废话。我想要阅读屏幕识别的代码，你的书提供了非常大的帮助！ -- Dattatray

> Hi, 《A byte of python》对于Python初学者来说真是一本不错的教材，你干得非常棒！我来自中国，是一名有4年的Java和C的经验的程序员。现在我想用pygtk做一些zim-wiki note project方面的事情。 我用6天的时间看完了这本书，现在我可以写Python的程序了。 谢谢你的贡献！ 请继续保持你的热情，并收下来自中国的祝愿！ -- Lee

> 我叫Isen，来自中国台湾, 是台湾国立大学电子工程系博士班的学生。我要谢谢你写了这么好的一本书。他不仅仅非常容易理解，而且对于Python初学者来说也足够完整。我阅读这本书是因为我要从事GNU Radio framework这方面的工作。你的书让我迅速的抓住了Python的核心思想，并且学会了Python编程。 我注意到了你并不介意读者给你发信息表示感谢，所以我再次谢谢你写了这么好的一本书。 -- Isen I-Chun Chao
>
> NASA甚至也使用这本书！在他们的喷气推进实验室Jet Propulsion Laboratory的深空网项目中使用。

学术课程

本书正在或曾经在多个院校作为教材使用：

- '编程语言原理'，课程在自由大学, 阿姆斯特丹Vrije Universiteit, Amsterdam
- '计算的基本概念'课程在加州大学戴维斯分校University of California, Davis
- '使用Python编程'课程在哈佛大学Harvard University
- '编程导论'课程在英国利兹大学University of Leeds
- '应用程序编程导论'课程在波士顿大学Boston University
- '气象学信息技术'课程在奥克拉荷马大学University of Oklahoma
- '地理数据处理'课程在密歇根州立大学Michigan State University
- '多代理语义网络系统'课程在英国爱丁堡大学University of Edinburgh
- '计算机科学与程序设计导论' at MIT OpenCourseWare
- '程序设计基础课程，利久不加纳大学, 斯洛文尼亚' -- Aleš Žiberna

协议

本书基于Creative Commons Attribution-ShareAlike 4.0 International License协议。

这意味着：

- 允许自由共享（例如复制）、分发以及传播本书
- 允许自由混合（例如改编）本书
- 允许商业目的自由使用

请注意:

- 请不要出售本书的电子或纸质版本，除非你在说明书中清晰而明显地声明，这些不是来

自本书原作者。
- 归属必须在序言以及文档的扉页以链接到的形式标明，并明确指出原始文本可以在此地址获得。
- 本书中提供的所有代码/脚本基于3-clause BSD License协议许可，除非额外声明。

开始阅读

你可以在线阅读本书。

购买本书

为了离线愉快阅读并支持本书的持续发展和改善，可以购买纸质印刷书籍。

下载

- PDF (for desktop reading, etc.)
- EPUB (for iPhone/iPad, ebook readers, etc.)
- Mobi (for Kindle)

你可以阅读原文查看原始的内容以便于更正、修订、翻译等等。

阅读本书的翻译

如果您对阅读或参与本书的翻译感兴趣，请看翻译。

继续阅读致敬

致敬

感谢Kalyan Varma和其他PESIT的管理者，他们带领我们认识了GNU/Linux和开源软件。

纪念Atul Chitnis，一位值得永久怀念的朋友。

感谢互联网的缔造者。这本书第一次出版是在2003年，到现在仍然非常受欢迎，感谢那些充满想象力的互联网缔造者，创造者了这么好的一个知识分享平台。

继续阅读前言

前言

Python可能是少数的几个既简单易学又功能强大的编程语言之一。对于初学者或者专家来说都非常适合，尤其是使用Python编程是一件非常快乐的事情。本书旨在帮助读者学习这门伟大的语言，让编程工作变的更加轻松。 这就是所谓的“工欲善其事必先利其器”。

读者对象

本书作为Python编程语言的指南或教程，主要面向初学者，同时对有经验的程序员也有帮助。

本书目的是，如果对于计算机，你只知道如何保存文本文件，那么你可以从本书学习Python。如果之前你有编程经验，那么你同样可以从本书学习Python。

如果您之前有过编程经验，你将对Python和你喜欢的编程语言之间的区别感兴趣——我高亮显示了这些区别。然而要提醒一点，Python将很快成为你最喜爱的编程语言！

官方网站

本书的官方网站为http://python.swaroopch.com/，你可以在线阅读全部的内容，下载最新的版本，或者购买纸质印刷书籍，也可以给我反馈。

要思考的一些事情

> 构建软件设计有两种途径：一种是足够简单以致明显没有缺陷，另一种是足够复杂以致没有明显缺陷。
>
> -- C. A. R. Hoare
>
> 人生的成功，专注和坚持比天才和机会更重要。
>
> -- C. W. Wendte

继续阅读关于Python

简介

Python是可以称得上即简单又功能强大的少有的语言中的一种。你将会惊喜地发现，专注于问题的解决方案而不是你正在使用的编程语言的语法以及结构，是多么容易。

官方对Python的介绍：

> Python是一个易于学习的、功能强大的编程语言。它具有高效的高级数据结构和能够简单有效地实现面向对象编程。Python优美的语法和动态类型，连同解释型特性一起，使其在多个平台的许多领域都成为脚本处理以及快速应用开发的理想语言。

在下一章，我将更详细地讨论这些特性。

名字背后的故事

> Python语言的发明人Guido van Rossum以BBC的喜剧《Monty Python's Flying Circus》给这个语言命名。他不是特别喜欢那些为了食物而杀死动物的蛇，这些蛇会用它们长长的身体缠绕住那些动物从而勒死它们。

Python的特点

简单

Python是一门简单而文字简约的语言。阅读优秀的Python程序感觉就像阅读英语，尽管是非常严格的英语。Python的这种伪代码特性是其最大强项之一，它可让你专注于解决问题的办法而不是语言本身。

容易学习

正如你即将看到的，Python非常容易上手。就像刚刚提到的，Python具有格外简单的语法。

免费开源

Python是一个FLOSS（自由/自由与开源软件）的例子。在一些简单的条款之下，你可以自由地分发这个软件的拷贝，阅读其源代码，修改它，或者将其一部分用到新的自由程序中。FLOSS是基于共享知识社区的概念，这是Python如此好的原因之一——它是由那些希望看到更好的Python的社区创建和不断改进的。

高级语言

当你使用Python编写程序时，你永远不需要担心低级细节，比如你的程序管理内存的使用等。

可移植

基于其开放源代码的特性，Python已经被移植（也就是使其工作）到许多平台。只要你足够小心，避免使用系统相关特性，你的所有Python程序都可以不加修改地运行在这其中任意平台。

你可以在Linux、Windows、FreeBSD、Macintosh、Solaris、OS/2、Amiga、AROS、AS/400、BeOS、OS/390、z/OS、Palm OS、QNX、VMS、Psion、Acorn RISC OS、VxWorks、PlayStation、Sharp Zaurus、Windows CE，甚至PocketPC平台上使用Python。

你甚至可以使用Kivy平台为iOS（iPhone、iPad）和Android创建游戏。

解释型

这需要一些解释。

使用编译型语言（像C或者C++）编写的程序，会由编译器使用一系列标志和选项，将源代码（如C或者C++）转换成一种电脑能够识别的语言（二进制代码，也就是0和1）。在运行程序时，链接器/载入软件将程序从硬盘复制到内存，然后开始运行。

换句话说，Python不需要编译成二进制代码。你只需从源代码直接运行程序。在内部，Python将源代码转换成一种称为字节码的中间格式，然后将其翻译你的计算机的机器语言，然后开始运行。事实上，这一切都让Python的使用更为简单，因为你不必担心程序的编译、保证恰当的库被链接和载入等等。这也使得你的Python程序更易于移植，因为你只需要复制你的Python程序到另外一台计算机，然后它就可以工作了！

面向对象

Python同时支持面向过程和面向对象编程。在面向过程语言中，程序围绕着过程或者函数（只不过是可重复使用的程序片段）构建。在面向对象语言中，程序围绕着对象（数据和功能的组合）构建。Python具有非常强大但是过于简洁的执行面向对象编程的方式，特别是相对于C++或者Java这种大型语言来说。

可扩展

如果你需要一段运行很快的关键代码，或者是想要编写一些不愿开放的算法，你可以使用C或C++完成那部分程序，然后从你的Python程序中调用。

可嵌入

你可以将Python嵌入到C/C++程序，让你的程序的用户获得"脚本化"的能力。

扩展库

Python标准库的确很大。它能够帮助你完成许多工作，包括正则表达式、文档生成、单元测试、线程、数据库、网页浏览器、CGI（公共网关接口）、FTP（文件传输协议）、电子邮件、XML（可扩展标记语言）、XML-RPC（远程方法调用）、HTML（超文本标记语言）、WAV（音频格式）文件、加密、GUI（图形用户界面）以及其它系统相关的代码。记住，只要安装了Python，所有这些都能做到。这叫做Python的"遥控器"哲学。

除了标准库，还有各式各样的其它高质量库，你可以在Python包索引找到它们。

小结

Python的确是一个激动人心的功能强大的语言。Python那种性能和特性的恰到好处的组合让使用Python编程既有趣又简单。

Python 3 vs 2

如果你不关心Python 2和Python 3的区别，可以跳过这一节。但是必须知道你正在使用的版本。本书使用Python 3完成。

一旦你充分地理解或学习使用了其中的一个，你可以很容易学到两个版本之间的区别，然后很容易的适应。困难的是学习编程和理解Python语言的核心，这是本书的目标。一旦你达到这个目标，你可以根据自己的情形很容易的使用Python 2或Python 3。

关于Python 2和Python 3的详细区别，请参考：

- Python 2的未来
- 将Python 2代码移植到Python 3
- 怎样书写能够同时运行在Python2 and 3中的代码
- 使用Python 3: 深度指南

程序员说了些什么

或许你会对顶尖的黑客，比如ESR，怎么看待Python感兴趣：

> Eric S. Raymond，是《The Cathedral and the Bazaar》的作者，也是发明开放源代码这一术语的人。他说，Python已经成为他最喜欢的编程语言。这篇文章给我第一次关注Python的真正灵感。
>
> Bruce Eckel，是著名的《Thinking in Java》和《Thinking in C++》的作者。他说，没有什么语言能比Python更能令他高效。他说，Python或许是唯一让程序员工作更简单的一个语言。请看完整的采访。
>
> Peter Norvig，是著名的Lisp的作者，Google搜索质量主管（感谢Guido van Rossum指出）。他说用Python编程就像是在写诗一样。他还说，Python一直是Google的主要部分。你可以通过查看Google Jobs验证这句话。这个页面上显示出，Python知识是招聘软件工程师的要求之一。

继续阅读安装

安装

本书所描述的"Python 3"，指的是Python 3.5.2或更高的版本。

在Windows上安装

访问[下载]https://www.python.org/downloads/最新版本。安装过程和其它基于Windows的软件类似。

警告

> 一定要选择 `Add Python 3.5 to PATH`

如果想要修改安装路径，点击 `Customize installation` , 再点击 `Next` ，键入 `C:\python35` 作为新的安装路径。

如果在安装的时候没有选择，那么请 `添加Python环境变量` . 这和 `Add Python 3.5 to PATH` 效果相同。

你可以选择为所有用户安装Python Launcher，这不是特别重要的选项。Launcher用来在不同的Python版本之间进行切换。

如果你指定的安装路径不正确，你需要修正他。否则，请参考 `在Windows命令行上运行Python` 。

提示: 对于有经验的程序员，如果你对Docker比较熟悉，请参考Python in Docker 和 Docker on Windows.

DOS提示符

如果你想要在Windows命名行，例如DOS提示符，使用Python，那么你需要正确设置PATH变量。

对于Windows 2000、XP、2003，点击控制面板---系统---高级---环境变量。在“系统变量”中点击PATH，选择编辑，然后在已有内容的最后部分添加;C:\Python35（请核实存在该文件夹，对于较新版本Python来说，文件夹的名字可能不同）。当然，要使用正确的目录名。

对于早期版本的Windows，打开C:\AUTOEXEC.BAT文件，添加一行“PATH=%PATH%;C:\Python35”（不含引号），然后重启系统。对于Windows NT，使用AUTOEXEC.NT文件。

对于Windows Vista:

- 点击“开始”，选择“控制面板”。

- 点击“系统”，在右侧您将看到“查看计算机基本信息”。
- 左侧是一个任务列表，其最后一项是“高级系统设置”，点击它。
- 显示“系统属性”对话框“高级”选项卡。点击右下角的“环境变量”按钮。
- 在下方标题为“系统变量”框中，滚动“Path”，点击“编辑”按钮。
- 按需修改路径。
- 重启系统。除非重启，Vista不会意识到系统路径变量的修改。

对于Windows 7/8/10:

- 在桌面上右击“计算机”，选择“属性”；或点击“开始”，选择“控制面板”---“系统和安全”---“系统”，点击左侧的“高级系统设置”，然后选择“高级”选项卡。点击底部的“环境变量”按钮，在下方的“系统变量”中找到PATH变量，选中它点击“编辑”。
- 在变量值的最后，追加;C:\Python35。 如果这个值是%SystemRoot%\system32;，它将变成%SystemRoot%\system32;C:\Python35。
- 点击“确定”完成。不需要重启。

在Windows命令行上运行Python

对于Windows用户，如果正确设置了PATH变量，你可以在命名行运行解释器。

要打开Windows终端，点击开始按钮，点击“运行”。在对话框输入cmd，按下回车键。

然后输入

```
python -V
```

确保没有错误。

在Mac OS X上安装

对于Mac OS X用户，使用Homebrew命令： `brew install python3` 安装。

通过按Command+Space键打开终端（打开Spotlight搜索），输入 `Terminal` 然后回车。之后，运行

```
python3 -V
```

确保没有错误。

在GNU/Linux上安装

对于GNU/Linux用户，使用你的Linux发行版的包管理器来安装Python 3，例如如果你使用的是Debian & Ubuntu: `sudo apt-get update && sudo apt-get install python3` 。

对于Linux用户，通过打开Terminal应用程序打开终端，或者按下Alt + F2，然后输入gnome-terminal。如果不成功，请参考文档或你所用Linux发行版的论坛。

一旦你完成安装，在shell运行python3 -V，在屏幕上你应该能够看到Python版本：

```
$ python3 -V
Python 3.5.2
```

提示：是shell的提示符，根据你电脑上的操作系统的设置会有所不同，因此我将使用$符号。

注意：在不同的电脑上输入信息可能会有所不同，取决于你安装的Python版本。

总结

现在开始，我们假设你已经在你的系统上安装好了Python 3。

接下来，我们将开始编写我们的第一个Python程序。

继续阅读第一步

第一步

现在,我们将看到在Python中如何运行一个传统的“Hello World”程序。这将教你如何写、保存和运行Python程序。

使用Python运行你的程序有两种方法——使用交互式解释器提示符或使用一个源文件。现在,我们将看到如何使用这两种方法。

使用解释器提示符

在您的操作系统中打开终端(如前面安装所述),然后，输入 `python3` 按回车键，打开Python提示符。

一旦你启动python 3,您应该看到'>>>",这被称为*Python*解释器提示符，你可以开始输入Python程序。

在Python解释器提示符下，输入

```
print("Hello World")
```

然后按回车键。您应该看到输出了单词“Hello World”。

当使用一个Mac OS X计算机，下面是你将看到的一个例子。Python软件的细节会根据你的电脑不同而有所不同，但从提示符(即从“>>>”开始)与操作系统无关，应该是相同。

```
> python3
Python 3.5.2 (default, Jan 14 2016, 06:54:11)
[GCC 4.2.1 Compatible Apple LLVM 7.0.2 (clang-700.1.81)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> print("Hello World")
Hello World
```

注意,Python让你的代码行立即输出了!你刚才输入的是一个Python语句。我们使用 `print` 输出(不出所料)你提供给它的任何值。在这里,我们提供的是文本“Hello World”,并立即打印到屏幕上。

如何退出解释器提示符

如果你正在使用一个Linux或Unix shell,您可以通过按下 `[ctrl - d]` 或输入 `exit()` (注意:记得包含括号, `()`)，然后输入回车键。

如果您使用的是Windows命令行提示符,按 `[ctrl - z]` 键再按回车键，退出解释器提示符。

选择一个编辑器

我们不能在每次想要运行一些东西的时候都要在解释器提示符下输入我们的程序，所以我们必须把它们保存为文件，这样我们可以任意次地运行我们的程序。

要创建我们的Python源文件,我们需要一个可以输入并保存它们的编辑软件。一个优秀的程序员的编辑器将使你写源代码文件的生活更容易。因此，选择一个编辑器确实至关重要。你必须选择一个编辑器，就像你选择要买的汽车一样。一个好的编辑器会帮助您很容易地编写Python程序，（就像一辆车可以让你）以一个更快和更安全的方式，让你的旅程更舒适，并且可以帮助你实现你的目标。

一个非常基本的需求是语法高亮显示，分别以不同的彩色显示你的Python程序所有的不同部分，以便您可以看到你的程序且使其运行可视化。

如果你不知道从哪里开始，我推荐可以在Windows、Mac OS X和GNU/Linux上使用的Visual Studio Code(简称VSCode)免费软件与Python插件(ext install python)。

如果您使用的是Windows，不要使用记事本——这是一个糟糕的选择，因为它不做语法高亮显示，而且更重要的是它不支持文字的缩进——之后我们在我们的例子中会看到，缩进是非常重要的。好的编辑器如Komodo Edit会自动地做到这一点。

如果你是一名有经验的程序员，那么你一定已经使用Vim或Emacs了。不用说，这是两个最强大的编辑器，使用它们来写你的Python程序，你会从中受益。就我自己而言，在我的大多数项目,甚至写一整本书都在用Vim。

从长远来看Vim或者Emacs是非常有用的，如果你愿意花时间去学习，那么我强烈建议你使用它们。然而,正如我之前提到的,初学者可以从PyCharm开始学习Python而不是编辑器。

再次重申，请选择一个适当的编辑器，它可以使编写Python程序更有趣和更容易。

VSCode

Visual Studio Code是一个免费的集成开发环境（IDE），你可以用它开发Python程序。

下载安装之后，在菜单中选择“查看”->“扩展”，然后输入 `python` ，选择由 `Don Jayamanne` 开发的Python扩展安装即可。



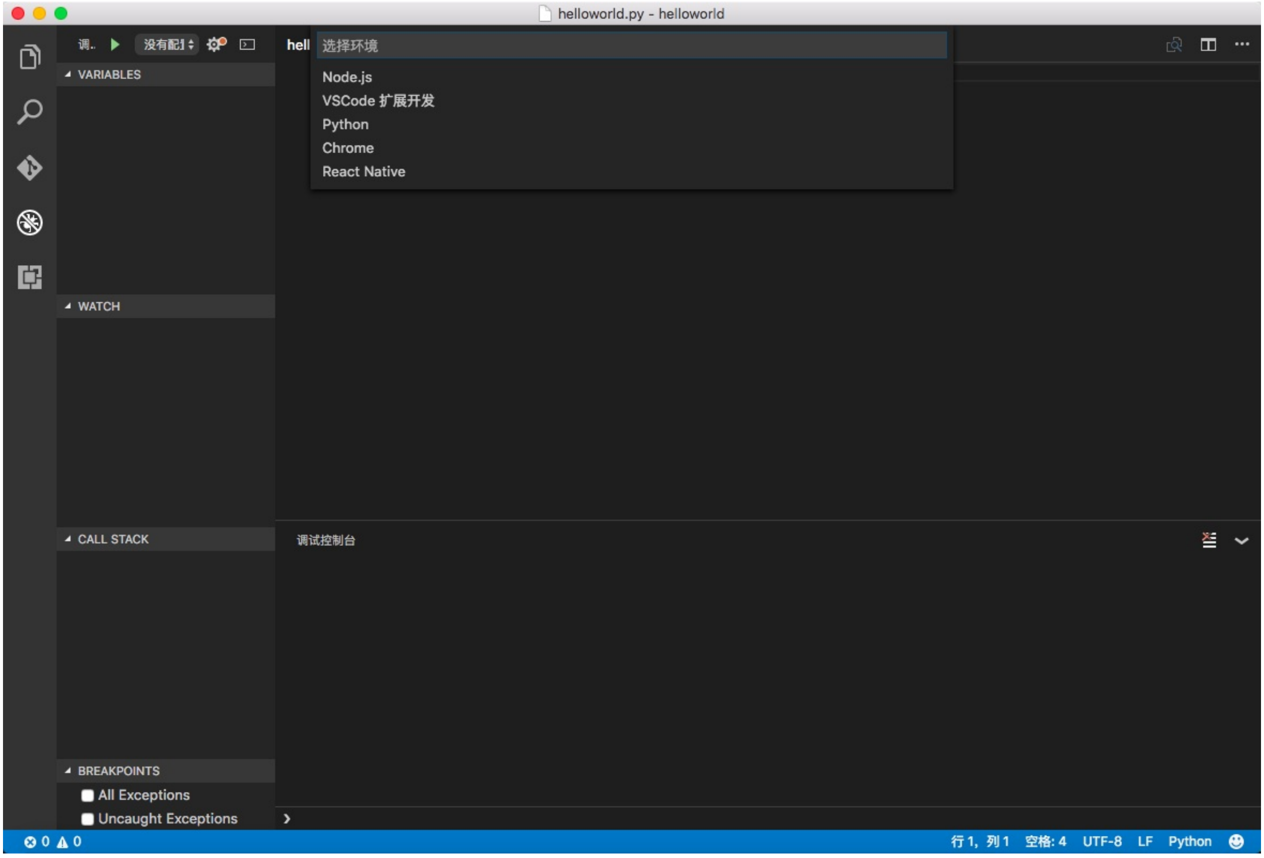
在VSCode中，新建项目就是新建一个文件夹，你可以使用操作系统的资源管理器新建文件夹，然后在当前文件夹下使用命令行 `code .` 启动VSCode。也可以在VSCode中使用“文件”->"打开...“打开这个文件夹。

打开文件夹之后，在左侧的资源管理器中右键单击，在弹出菜单中选择“新建文件”，然后输入 `helloworld.py` ，点击回车即可。

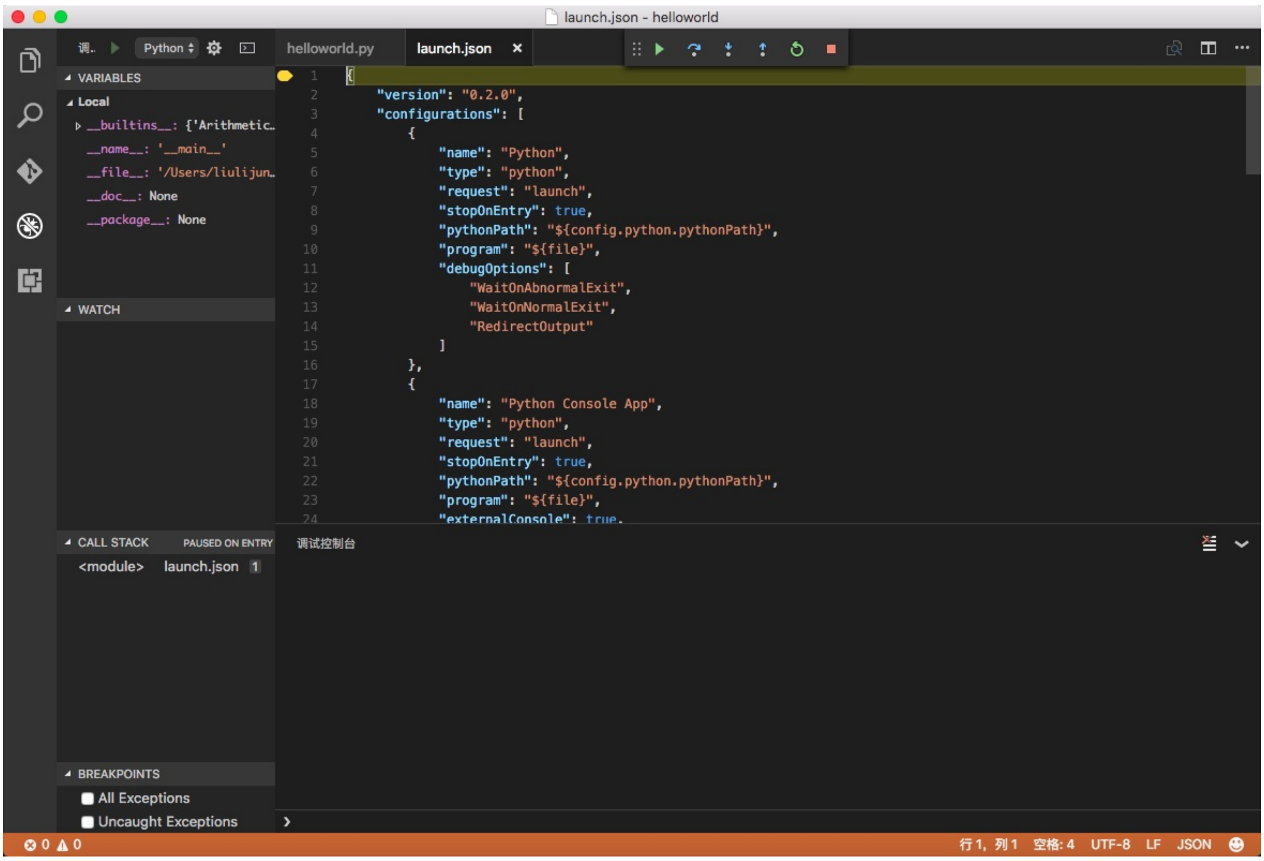
输入以下代码：

```
print("hello world!")
```

点击 `[Ctrl+S]` 保存文件之后，选择最左侧的 `调试` 视图，然后点击上面的绿色小三角，运行 `helloworld.py` 程序。此时VSCode会弹出一个列表让你选择环境，在这里选择 `Python` 即可。



再次点击绿色的小三角按钮，会看到程序没有立即运行，而是进入了调试状态，在中间出现了调试的对话框，点击那个绿色的小三角按钮即可运行程序，看到 `hello world!` 的输出。



之所以要再点击一次，是因为VSCode的Python插件默认的启动配置为：

```
"stopOnEntry": true,
```

可以修改为false，这样以后点击“运行”的时候就直接执行程序了。

Vim

1. 安装Vim
    - Mac OS X用户应该通过HomeBrew安装 `macvim` 软件 。
    - Windows用户应该在Vim网站下载exe安装文件。
    - GNU/Linux用户一般情况下可以直接使用 `vim` 。
2. 你可以安装jedi-vim这个插件为vim增加自动完成的功能。
3. 安装对应的 `jedi` python包： `pip install -U jedi`

Emacs

1. I安装Emacs 24+.
    - Mac OS X用户从http://emacsformacosx.com获得emacs
    - Windows用户从http://ftp.gnu.org/gnu/emacs/windows/下载
    - GNU/Linux用户根据不同的发行版获得对应的emacs软件，比如Debian和Ubuntu用户可以安装 `emacs24` 软件包
2. 安装ELPY

使用一个源文件

现在让我们回到编程。每当你学习一种新的编程语言时，有一个传统，你编写和运行的第一个程序是“Hello World”程序——当你运行它时，它所做的只是说“Hello World”。正如Simon Cozens(神奇的"Beginning Perl"的作者)所说，这是“向编程神祈求帮你更好学习语言的传统咒语。”

开始你选择的编辑器，输入以下程序并将其保存为“hello.py’。

如果你使用VSCode编辑器，点击“文件”->“新建文件”,输入下行:

```
print('Hello World')
```

在VSCode编辑器，选"文件”->“保存”保存文件。

你应将文件保存在哪里？你知道位置的任何文件夹。如果你不明白这是什么意思，创建一个新文件夹，并使用该位置保存和运行你所有的Python程序:

- C:\py 在Windows上
- /tmp/py 在Linux上
- /tmp/py 在Mac OS X上

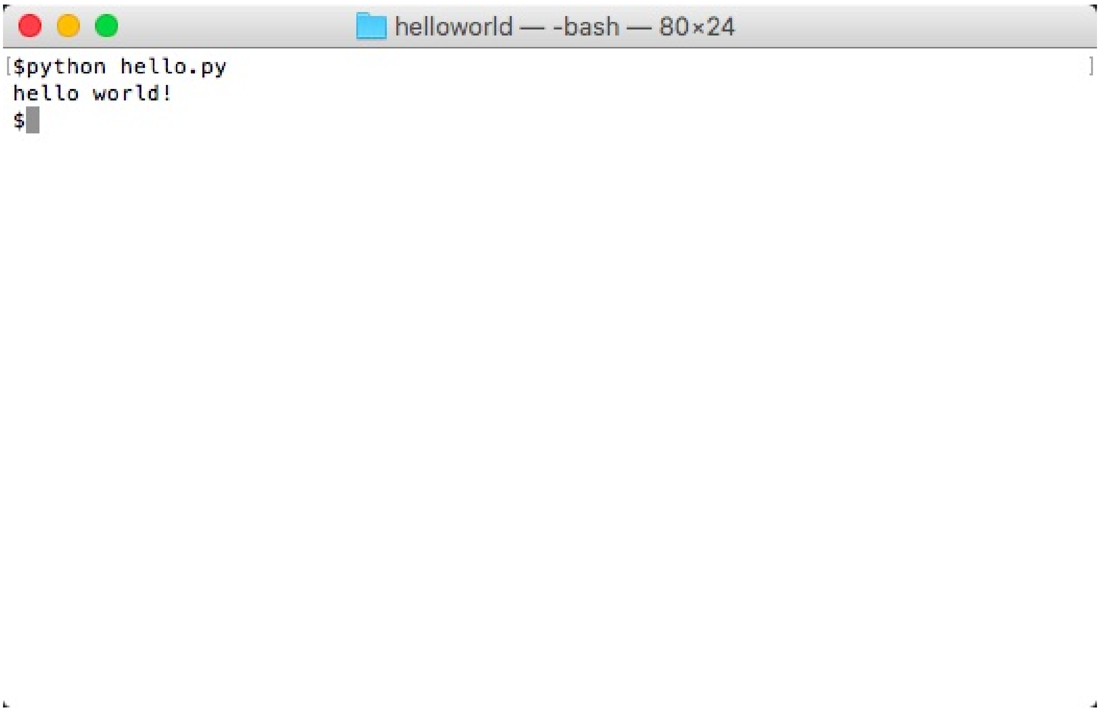
使用'mkdir'命令在命令行创建一个文件夹，例如,“mkdir /tmp/py”。

重要提示：确保你给它的文件扩展名是.py，例如，“foo.py”。

运行你的Python程序:

1. 打开一个命令行窗口。
2. 进入你刚才保存文件的目录，例如：`cd /tmp/py`。
3. 键入`python hello.py`运行python程序，输入如下：

```
$ python hello.py
hello world
```



如果你得到了如上所示有输出，祝贺你!——你已经成功地运行了你的第一个Python程序。您已经成功地越过学习编程最难的部分——开始你的第一个程序!

如果你得到了一个错误，请完全输入如上所示程序，再次运行这个程序。注意，Python是区分大小写的，即“print”并不等于“Print”——注意，前者是小写字母“p”和后者是大写字母“P”。同样，确保每一行的第一个字母之前没有空格或制表符——之后我们将明白为什么这很重要。

它是如何工作的

Python程序是由语句组成，在我们的第一个程序中，我们只有一个语句，在这个语句中，我们调用“print”函数，它只是打印文本“Hello World”。

获得帮助

如果您需要快速获取任何的Python函数或语句的信息，那么您可以使用内置的“help”(帮助)功能。这是非常有用的，尤其是当使用翻译提示符时，例如，运行‘help(print)”——这将显示print函数的帮助——用于打印东西到屏幕上。

注意：按q退出帮助

类似地,您可以获得Python中几乎任何事情的信息，使用“help()”去学习更多关于使用“help”本身的信息!

如果你需要获取操作符，如“return”的帮助，那么你只需要把这些放到引号内部，如“help('return')，所以，对于我们试图要做的事情，Python并不感到困惑。

总结

现在，你可以自由自在地编写、保存和运行Python程序了。

既然你是一名Python用户，让我们学习一些Python概念。

继续阅读基础

基础

只是打印“Hello World”是不够的，是吗？你想要做的不仅仅是这些，你想带一些输入，操纵它，得到些输出。在Python中，我们使用常量和变量可以实现这些。在这一章，我们还将学习一些其他的概念。

注释

注释是#符号右边的任何文字，它对读程序的人有很大用处。

例如：

```
print('hello world') # 注意：print是一个函数
```

或者:

```
# 注意：print是一个函数
print('Hello World')
```

在你的程序中，你尽可能使用有用的注释:

- 解释设想
- 解释重要决策
- 解释重要细节
- 解释你在试图解决的问题
- 解释程序中你在努力克服的问题，等等。

代码告诉你怎样做，注释告诉你这样做的原因。

这对你的程序的读者是有用的，他们可以很容易地理解程序做什么。记住，六个月后这个人可以是你自己！

字面常量

字面常量的一个例子是一个数字就像 `5` 、 `1.23` ，或一个字符串像“这是一个字符串”或“它一个字符串!”。它被称为字面，因为它是字面的——你使用它的字面值。数字 `2` 总是代表本身并没有什么其他的——它是一个常数，因为它的值是不能改变的，因此，所有这些被称为字面常量。

数字

数字主要有两种类型--整型和浮点数。

`2` 是整数的一个例子，它只是一个完整的数。

浮点数(或简称为浮点)的例子有 `3.23` 和 `52.3E-4` 。符号E表示10的次方。在这种情况下， `52.3E-4` 的意思是52.3的-4次方。

> 有经验的程序要注意： Python中没有单独的long(长)整型。int(整型)可以是任意大小的整数。

字符串

字符串是字符的一个序列，字符串通常只是一串单词。

在你写的每一个Python程序中都要使用字符串，因此要注意以下部分：

单引号

你可以使用单引号例如'Quote me on this'指定字符串。所有的空白，例如空格和制表符都按原样保留。

双引号

在双引号中的字符串和在单引号中的字符串工作起来完全一样。例如"What's your name?"

三重引号

您可以使用三重引号-("""或''')指定多行字符串。在三重引号中您可以自由使用单引号和双引号。例如:

```
'''This is a multi-line string. This is the first line.
This is the second line.
"What's your name?," I asked.
He said "Bond, James Bond."
'''
```

字符串是不可改变的

这意味着，一旦您已经创建了一个字符串，你就不能改变它。虽然这看起来似乎是一件坏事，但它真不是(坏事)。在我们后面看到的各种程序中，将会明白这不是一个限制。

> **C/C++**程序员要注意： 在Python中没有单独的“char”(字符型)数据。这里没有眞正的需要它，我相信你不会想念它。
>
> **Perl/PHP**程序员要注意： 记住，单引号字符串和双引号字符串是相同的——他们不以任何方式不同。

格式方法

有时，我们可能想要从其它信息构建字符串。这就是“format()”方法有用之处。

保存下面几行到文件"str_format.py"中:

```
age = 20
name = 'Swaroop'

print('{0} was {1} years old when he wrote this book'.format(name, age))
print('Why is {0} playing with that python?'.format(name))
```

输出结果为:

```
python str_format.py
Swaroop was 20 years old when he wrote this book
Why is Swaroop playing with that python?
```

它是如何工作的:

一个字符串可以使用特定的格式，随后调用format方法，用format方法替代那些使用适当参数的格式。

观察使用第一处，我们使用“{0}”对应于变量‘name’,这是format(格式)方法的第一个参数。类似的,第二个格式是“{1}”对应的“age”,这是格式方法的第二个参数。注意，Python从0开始计数，这意味着第一位置的索引是0，第二个位置的索引是1，等等。

注意，我们可以使用字符串的连接

```
name + 'is' + str(age) + 'years old'
```

实现同样的目的，但这非常讨厌、容易出错。第二，在这种情况下，通过format方法自动转换为字符串，而不是显式地转换为需要的字符串。第三，当使用的format 方法，我们可以改变消息，而无需处理使用的变量，反之亦然。

还要注意，这些数字(索引)都是可选的，所以你也可以写成:

```
age = 20
name = 'Swaroop'

print('{} was {} years old when he wrote this book'.format(name, age))
print('Why is {} playing with that python?'.format(name))
```

这将给与前面的程序相同的输出。

Python在format方法中做的是，用每个参数值替代规格的地方。这里有更详细的规格，如:

```
# decimal (.) precision of 3 for float '0.333'
print('{0:.3f}'.format(1.0/3))
# fill with underscores (_) with the text centered
# (^) to 11 width '___hello___'
print('{0:_^11}'.format('hello'))
# keyword-based 'Swaroop wrote A Byte of Python'
print('{name} wrote {book}'.format(name='Swaroop', book='A Byte of Python'))
```

输出为:

```
0.333
___hello___
Swaroop wrote A Byte of Python
```

既然我们正在讨论格式化，大家应该也已经注意到了： `print` 函数每次都会在输出的最后增加一个回车，这保证了我们重复调用 `print` 函数每次都能输出在不同的行。如果想要去掉这个回车，可以给 `print` 函数一个 `end` 参数并设置为空即可，例如:

```
print('a', end='')
print('b', end='')
```

输出是:

```
ab
```

或者你也可以给 `end` 参数指定为空格：

```
print('a', end=' ')
print('b', end=' ')
print('c')
```

输出为:

```
a b c
```

转义字符

假设你想要一个带单引号 (`'`)的字符串，你会怎么写呢？例如你想输出 `"What's your name?"` 。 你不能写成'What's your name?', 因为python会搞不清字符串是从哪开始又是从哪结束。所以你需要告诉python字符串中间的单引号并不是字符串的结束标志。 利用转义字符可以做到这点。 将单引号替换成 `/'` - 注意反斜杠，这样字符串就变成 `'What/'s your name?'` 了.

另一个办法是使用双引号 `"What's your name?"` 。 不过当你需要双引号的时候和单引号的情况类似，必须加上反斜杠 `/"` , 而反斜杠也一样必须表示成 `//` 。

如果你需要一个双行字符串呢? 一个办法是使用前面提到的三引号或者使用换行的转义字符 `\n` 开始新的一行，例如:

```
'This is the first line\nThis is the second line'
```

还有一个有用的转义字符 `\t` 表示tab, 转义字符太多了我只说了常用的。

另一个值得注意的地方是在一个字符串末尾的反斜杠表示这个字符串将被合并到下一行, 例如:

```
"This is the first sentence. \
This is the second sentence."
```

上面的字符串等价于

```
"This is the first sentence. This is the second sentence."
```

原始字符串

如果你不希望python对字符串进行特别处理, 不希望使用转义字符, 可以为字符串加上前缀r或R，这样的字符串就是原始字符串。例如：

```
r"Newlines are indicated by \n"
```

> 正则表达式用户请注意
>
> 永远使用原始字符串编写正则表达式, 否则会需要大量的反斜杠, 例如反向引用可以表示为 `'\\1'` 或 `r'\1'` .

变量

仅仅使用字面常量会很快变得无聊——我们需要某种方式存储任何信息，同样操作它们。这就要引入变量。变量是它的名字所指示的东西——他们的值会有所不同。例如，你可以用变量存储任何值。变量是你电脑中存储信息的内存的一部分。不像字面常量，您需要某种方法来访问这些变量，因此你要给出他们的名字。

标识符命名

变量是标识符的一个示例。标识符是用来识别一些东西的名字。这里有一些你必须遵循的标识符命名规则:

- 标识符的第一个字符必须是字母表中的一个字母(大写或小写的ASCII或Unicode字符)或下划线(`_`)。
- 标识符名称的其余部分可以包含字母(大写或小写的ASCII或Unicode字符)、下划线(`_`)或数字(0 - 9)。
- 标识符的名称都是区分大小写的。例如, `myname` 和 `myName` 不相同。注意前者中 `n` 小写和后者中 `N` 大写。
- 有效的标识符名称的例子有： `i` , `myname` , `name23` 。无效的标识符名字的例子有： `2things` , `this is spaced out` , `my-name` 和 `>a1b2_c3` 。

数据类型

变量可以保存不同的被称为数据类型的数值。基本类型是数字和字符串，我们已经讨论了。在后面的章节中，我们将看到如何使用类创建自己的类型。

对象

记住，Python把程序中使用的任何东西作为一个对象。在常识中这意味着，我们说‘对象’，而不是说‘一些东西’,

> 面向对象编程的用户要注意：一切东西都是对象，包括数字、字符串和函数，在这个意义上讲，Python是坚定的面向对象的。

现在，我们将看到如何使用变量以及字面常量。保存下面的示例，运行这个程序。

怎样写Python程序

从此以后，保存和运行一个Python程序的标准过程如下:

1. 打开选择的编辑器，例如VS Code。

2. 输入例子中给出的程序代码。
3. 用提到的文件名保存为文件。
4. 在命令行使用python program.py命令运行程序。

例如: 使用变量和常量

Filename : var.py

```
# Filename : var.py
i = 5
print(i)
i = i + 1
print(i)

s = '''This is a multi-line string.
This is the second line.'''
print(s)
```

输出结果为:

```
5
6
This is a multi-line string.
This is the second line.
```

它是如何工作的

下面介绍这个程序如何工作的。首先，我们使用赋值运算符(`=`)为变量 `i` 指定了字面常量值 `5` ,这一行叫做一个声明，因为它指出应该做一些事情，在这个例子中，我们将名为 `i` 的变量与值 `5` 相联系。接下来,我们使用 `print` 函数打印 `i` 的值，不出所料，恰恰打印变量的值到屏幕上。

然后，我们给存储在变量 `i` 中的值加 `1` 后，再存回到 `i` 中。然后，我们把它打印出来，我我们想的一样，我们得到值 `6` 。

同样，我们为字符串变量 `s` 指定了字面常量字符串，然后打印它。

> 静态语言程序员应注意 变量的使用只是通过给他们指定一个值。不需要/不使用声明或数据类型定义。

逻辑行与物理行

物理行是当你写程序时看到的一行。逻辑行是*Python* 看到的一个单独语句。Python默认一个物理行为一个逻辑行。

一个逻辑行是一个语句，像 `print('Hello World')` ——如果它本身在一行上（像你在一个编辑器中看到的），那么，它也是一个物理行。

默认情况下，Python鼓励一行写一个语句的用法，这使代码更可读。

如果您想要在一个物理行列举多个逻辑行，那么您必须使用一个表示逻辑行/语句结束的分号 `;` 显式地指明。例如：

```
i = 5
print(i)
```

与

```
i = 5;
print(i);
```

等效。

同样可写成：

```
i = 5; print(i);
```

甚至是

```
i = 5; print(i)
```

然而，我强烈建议你坚持在每一个物理行编写一个最大的逻辑行。这就是你永远都不要使用分号。事实上，我从未使用，甚至在python程序中从来没有见过一个分号。

这个观念是很有用的，还有一种情况：如果你有一个长代码行，你可以通过使用反斜杠把它分解为多个物理行。这是被称为显式行连接:

```
s = '这是一个字符串。 \
这是字符串的继续。'
print(s)
```

输出结果为：

```
这是一个字符串。这是字符串的继续。
```

同样的，

```
i = \
5
```

与

```
i = 5
```

相同

有时有一种隐含的假设，您不需要使用一个反斜杠。在这种情况下，逻辑行有一个开始圆括号、开始方括号或开始花括号，但不是一个结束的括号。这被称为隐式连接。当我们在以后的章节——编写程序使用列表时，你可以看到它的作用。

缩进

在Python中的空白是重要的。实际上，在一行开始的空格是重要的。这被称为缩进。在逻辑行开头的前导空白(空格和制表符)用于确定逻辑行的缩进级别，它用于依次确定语句的分组。

这意味着一起的语句必须有相同的缩进。每一个这样的语句组被称为块。在后面的章节，我们将看到的块是何等重要的例子。

你应该记住的一件事是，错误的缩进可以产生错误。例如:

```
i = 5
 print('值是 ', i) # 错误! 注意在行的开头有一个空格
print('重复，值是 ', i)
```

当运行它时，将会发生下面的错误：

```
  File "whitespace.py", line 4
    print('Value is ', i) # Error! Notice a single space at the start of the line
    ^
IndentationError: unexpected indent
```

请注意，这里第二行的开头有一个空格。这个错误表明：Python告诉我们程序的语法是无效的，即程序写的不正确。这意味着，你不能任意开始语句中的新块(当然，除了默认的主块，您一直已经使用的)。您可以使用新块的情况，将在后面的章节详细，如"控制流"。

> 如何缩进 缩进只使用4个空格，这是Python官方推荐的标准。好的编辑器会为你自动这样做。确保你使用一致的数量的缩进空格，否则你的程序将显示错误。
>
> 静态语言程序员应注意 Python为块总是使用缩进，从来不用花括号。运行 `from __future__ import braces` 可以了解更多。

小结

现在，我们已经经历了许多细节,我们可以转到更有趣的东西，如控制流语句。一定要熟悉这一章你所读的。

继续阅读操作符和表达式

操作符和表达式

你编写的大多数语句（逻辑行）都将包含表达式。一个表达式的简单例子是 `2+3` 。一个表达式可分解成操作符和操作对象。

操作符的功能是做一些事，通过符号（如 `+` ）或特定的关键字表示。操作符需要一些数据来操作，这些数据被你作操作对象。在这个例子中 `2` 和 `3` 是操作对象。

操作对象

我们将简单地看一下操作符和它的用法：

注意，您可以使用交互式解释器计算例子中给出的表达式。例如,为了测试表达式 `2 + 3` ，使用交互式Python解释器提示符:

```
>>> 2 + 3
5
>>> 3 * 5
15
>>>
```

以下是一些操作符的简介：

- `+` (加号)
    - 两个对象相加
    - `3 + 5` 得 `8` ， `'a' + 'b'` 得 `'ab'` 。
- `-` (减号)
    - 给出一个数减去另一数的差；如果缺少第一个操作数，它默认为是0。
    - `-5.2` 得到一个负数， `50 - 24` 得 `26` .
- `*` (乘法)
    - 给出两个数的乘积或返回重复多次的字符串。
    - `2 * 3` 得 `6` . `'la' * 3` 得到 `'lalala'` .
- `**` (幂)
    - 返回x的y次幂
    - `3 ** 4` 得 `81` (也就是 `3 * 3 * 3 * 3`)
- `/` (除法)
    - x除以y

  - `13 / 3` 得 `4.333333333333333`.
- `//` (整除)
  - 返回最大的整数商
  - `13 // 3` 得 `4`
  - `-13 // 3` 得 `-5`
- `%` (取模)
  - 返回除法的余数
  - `13 % 3` 得 `1`. `-25.5 % 2.25` 得 `1.5`.
- `<<` (向左移位)
  - 数字向左移动指定位数。(在内存中每个数字由比特或二进制数表示，例如：0和1）。
  - `2 << 2` 得 `8`. `2` 用二进制表示为 `10` 。
  - 左移两位得到 `1000` ，它表示数字 `8` 。
- `>>` (向右移位)
  - 数字向右移动指定位数。
  - `11 >> 1` 得 `5`.
  - `11` 用二进制表示为 `1011` ，向右移动1位后得到二进制 `101` ，表示数字 `5` 。
- `&` (按位与)
  - 数字的按位与
  - `5 & 3` 得 `1` 。
- `|` (按位或)
  - 数字的位相或
  - `5 | 3` 得 `7` 。
- `^` (按位异或)
  - 数字的位相异或
  - `5 ^ 3` 得 `6` 。
- `~` (按位求反)
  - x的位求反结果为-(x+1)
  - `~5` 得 `-6` 。详见 http://stackoverflow.com/a/11810203
- `<` (小于)
  - 返回x是否小于y。所有的比较运算符返回 `True` 或 `False` 。注意这些名字的大小写。
  - `5 < 3` 返回 `False` 而 `3 < 5` 返回 `True`.
  - 比较运算符可以任意连接： `3 < 5 < 7` 返回 `True`.
- `>` (大于)
  - 返回x是否大于y

  - `5 > 3` 返回 `True` 。如果操作对象都是数字，它们首先转换为普通型，否则，将返回 `False` 。
- `<=` (小于等于)
  - 返回x是否小于等于y
  - `x = 3; y = 6; x <= y` 返回 `True` .
- `>=` (大于等于)
  - 返回x是否大于等于y
  - `x = 4; y = 3; x >= y` 返回 `True` 。
- `==` (等于)
  - 比较操作对象是否相等
  - `x = 2; y = 2; x == y` 返回 `True` .
  - `x = 'str'; y = 'stR'; x == y` 返回 `False` .
  - `x = 'str'; y = 'str'; x == y` 返回 `True` .
- `!=` (不等于)
  - 比较操作对象是否不相等
  - `x = 2; y = 3; x != y` 返回 `True` .
- `not` (逻辑非)
  - 如果 x 是 `True` ,它返回 `False` 。如果 x 是 `False` ，它返回 `True` 。
  - `x = True; not x` 返回 False.
- `and` (逻辑与)
  - 如果x是 `False` , `x and y` 返回 `False` ，否则它返回y的值。
  - `x = False; y = True; x and y` 返回 `False` ，因为 x 为假。在这种情况下，Python将不计算y，因为它知道 `and` 左边表达式是 `False` ，这意味着整个表达式将为 `False` ，而不论其它值为什么。这叫做求值捷径。
- `or` (逻辑或)
  - 如果 x 为 `True` , 它返回真，否则它返回y的值。
  - `x = True; y = False; x or y` 返回 `True` 。求值捷径这也适用。

数学操作和赋值的快捷方式

我们经常会对一个变量进行数学操作，然后将操作的结果返回给最初的变量。这些表达式有一个快捷方式:

```
a = 2
a = a * 3
```

可以写成：

```
a = 2
a *= 3
```

注意：将 `var = var operation expression` 写成 `var operation= expression` 。

运算顺序

如果你有一个表达式如 `2 + 3 * 4`，是先做加法还是先做乘法呢？我们的高中数学告诉我们，应该先做乘法。这意味着乘法操作符比加法操作符具有更高的优先级。

下面的表给出了Python运算顺序的优先表，从最低（最小约束力）到最高(最高约束力)。意思是说，在给定的表达式中，Python将按照自下而上的顺序，首先计算优先表下方的表达式。

下面的表取自Python参考手册，是为了提供完整性。为了显式地指定优先级，更好的做法是使用圆括号组织运算符和操作对象。这可使程序更加可读。详见下面更改运算顺序。

- `lambda` ：Lambda表达式
- `if - else` ：条件表达式
- `or` ：逻辑或
- `and` ：逻辑与
- `not x` ：逻辑非
- `in, not in, is, is not, <, <=, >, >=, !=, ==` ：比较、成员检测、相等检测
- `|` ：按位或
- `^` ：按位异或
- `&` ：按位与
- `<<, >>` ：移位
- `+, -` ：加和减
- `*, /, //, %` ：乘法，除法，浮点除和余数
- `+x, -x, ~x` ：正，负，按位非
- `**` ：乘方
- `x[index], x[index:index], x(arguments...), x.attribute` ：索引，切片，函数调用，属性引用
- `(expressions...), [expressions...], {key: value...}, {expressions...}` ：显示Binding或元组，显示列表， 显示字典

我们没有遇到的操作符将在后面的章节解释。

上表中在同一行列出的操作符具有相同优先级。例如， `+` 和 `-` 具有相同的优先级。

改变运算顺序

为使表达式更具可读性，我们可以使用圆括号。例如 `2 + (3 * 4)` 肯定比需要知道操作符运算优先级的 `2 + 3 * 4` 更容易理解。与其他方面一样，应该合理使用括号不应该冗余(不要过分使用),如 `(2 + (3 * 4))` 。

使用括号有一个额外的优势——它帮助我们更改运算顺序。例如，如果您想要在一个表达式中加法在乘法之前运算，那么你可以这样写 `(2 + 3) * 4` 。

结合性

操作符通常从左到右。这意味着具有相同优先级的操作符从左到右的方式计算。例如 `2 + 3 + 4` 计算为 `(2 + 3) + 4` 。

表达式

例子 (保存为expression.py):

```
length = 5
breadth = 2

area = length * breadth
print('Area is', area)
print('Perimeter is', 2 * (length + breadth))
```

输出:

```
D:> python expression.py
Area is 10
Perimeter is 14
```

它是如何工作的

矩形的长度和宽度以同样的名字存储在变量中，在表达式的帮助下，我们使用这些计算矩形的面积和周长。我们存储表达式 `length * breadth` 的结果在变量area中，然后使用print函数打印它。在第二种情况下,在打印函数中我们直接使用表达式 `2 * (length + breadth)` 的值。

同样要注意，Python完美打印是如何输出的。即使我们没有在 `'Area is'` 和变量 `area` ,之间指定一个空间，Python为我们得到一个干净漂亮的输出，而且这种方式使用程序的可读性更强(因为我们不需要担心为输入我们在字符串中间使用的空格)。这只是让Python程序员的生活更轻松的一个例子。

小结

我们已经看到了如何使用操作符，操作对象和表达式——这是任何程序的基石。接下来,我们将看到在使用语句的程序中如何利用这些。

继续阅读控制流

控制流

在程序中，到现在为止，我们看到，一直有一系列的语句被Python以由上而下的顺序如实地执行。如果你想改变它的流程，它会如何工作呢？例如，你想让程序作出一些决定，而且不同的情况做不同的事情，例如，根据一天的时间不同，打印 `早上好` 或 `晚上好` ？

正如你可能已经猜到的，这要通过使用控制流语句。在Python中有三个控制流语句-- `if` , `for` 和 `while` 。

if 语句

if 语句是用来检查一个条件:如果条件为真,我们运行一个语句块(你为if块)，否则我们执行另一个语句块(称为else块)。else语句是可选的。

例如 (保存为 if.py):

```
number = 23
guess = int(input('请输入一个整数: '))

if guess == number:
    # 新块从这里开始
    print('恭喜，你猜对了。')
    print('(但你没有获得任何奖品！)')
    # 新块在这里结束
elif guess < number:
    # 另一个块
    print('不对，你猜的有点儿小')
    # 在一个块中你可以做你想做的任何事...
else:
    print('不对，你猜的有点大')
    # 你猜的数比number大时才能到这里

print('完成')
# if语句执行完后，最后的语句总是被执行
```

输出:

```
$ python if.py
请输入一个整数: 50
不对，你猜的有点儿大
完成

$ python if.py
请输入一个整数: 22
不对，你猜的有点儿小
完成

$ python if.py
请输入一个整数: 23
恭喜，你猜对了。
 (但你没有获得任何奖品！)
完成
```

它是如何工作的:

在这个程序中，我们获取来自用户的猜测，并检查这个数是否是我们设定的数。我们给变量number设置我们想要的任何整数，比如 `23` 。然后，我们使用 `input()` 函数获取用户的猜的数。函数是可重用的程序块。我们在下一章中会阅读关于它们的更多东西。

我们给内置的 `input` 函数提供一个字符串，该函数将其打印到屏幕上并等待用户输入。一旦我们输入一些东西并按下回车键， `input()` 函数把我们的输入作为一个字符串返回。然后，我们使用 `int` 将这个字符串转换为整数，然后将其存储在变量 `guess` 中。实际上， `int` 是一个类，但现在所有你需要知道的是，您可以使用它来将一个字符串转变为一个整数(假设文本中的字符串包含一个有效的整数)。

接下来,我们比较用户猜的数和我们选择的数，如果他们相等，我们打印一条成功的消息。注意，我们使用缩进级别告诉Python语句属于哪个块。这就是为什么缩进P在ython中是如此重要。我希望你坚持"一致的缩进"的规则，好吗?

注意， `if` 语句在最后有一个冒号——我们指示Python一个语句块将跟随其后。

然后,我们检查猜的数是否小于这个数字，如果是，我们通知用户，他们猜的数必须比那个数稍高。我们这里使用的是 `elif` 子句，实际上将两个相关的 `if else-if else` 语句组合为一个语句 `if-elif-else` ，这使程序更简单且减少所需要的缩进。

`elif` 和 `else` 语句也必须在逻辑行结束时有一个冒号，后跟相应的语句块(当然要通过适当的缩进)

你可以在 `if` 语句的 `if` 块中有另一个 `if` 语句——这称为 `if` 语句嵌套。

记住， `elif` 和 `else` 部分是可选的。一个最小的有效的 `if` 语句是：

```
if True:
    print('是的，它为真')
```

在Python执行完成完整的if语句以及相关的 `elif` 和 `else` 子句，它移动到 `if` 包含语句的块中下一个语句块。在本例中，它是主要的块(程序开始执行的地方)，接下来的语句是 `print('完成')` 。在这之后，Python将看到程序的结尾，并简单的完成。

尽管这是一个非常简单的程序，但你应该注意，我已经指出很多东西。所有这些都是相当的直截了当(对那些有C/C++背景的人是惊人的简单)。最初，你需要意识到所有这些事情，但经过一些练习，对它们你将感到舒服，自然将是你所有的感觉。

> C/C++程序员需要注意 在Python中没有 `switch` 语句。您可以使用一个 `if..elif..else` 语句做同样的事(和在某些情况下,使用词典去做更快速)

while语句

只要条件为真， `while` 语句允许您多次执行一个语句块。 `while` 语句是被称为循环语句的一种。while语句可以有一个可选的 `else` 子句。

例如 (保存为while.py):

```
number = 23
running = True

while running:
    guess = int(input('输入一个整数 : '))

    if guess == number:
        print('恭喜，你猜对了。')
        # 这使while循环停止
        running = False
    elif guess < number:
        print('不对，你猜的有点儿小。')
    else:
        print('不对，你猜的有点儿大。')
else:
    print('while循环结束。')
    # 在这做你想做的任何事

print('完成')
```

输出:

```
$ python while.py
输入一个整数 : 50
不对，你猜的有点儿大。
输入一个整数 : 22
不对，你猜的的点儿小。
输入一个整数 : 23
恭喜，你猜对了。
while循环结束。
完成
```

它是如何工作的：

在这个程序中，我们还是玩猜谜游戏，但优点在于，允许用户一直猜直到他猜对——每次猜测不需要重复运行该程序，正如我们在前一节中所做的。这演示了如何恰当的使用 `while` 语句。

我们移动 `input` 和 `if` 语句到 `while` 循环中，在 `while` 循环前，设置变量 `running` 为 `True` 。首先，我们检测变量 `running` 是否为 `True` ，然后往下执行相应的*while*块。在这个块执行完后，再检测条件，在这里是变量 `running` 为真，我们再次执行while块，否则，我们执行可选的else块，然后执行下面的语句。

当 `while` 循环的条件变为 `False` 时--这也可能发生在条件检测时的第一次，执行 `else` 块。如果你使用 `break` 语句退出循环的话，那么 `else` 子句就不会被执行，否则当 `while` 循环的条件变为 `False` 时就会执行。

在这里 `True` 和 `False` 被称为布尔类型，你可以认为它们分别相当于值 `1` 和 `0` 。

> C/C++程序员注意： 记住， while循环可以有else子句。

`for` 循环

`for..in` 语句是另一个循环语句，它迭代一个对象的序列，例如经历序列中的第一项。在后面的章节，我们将会看到更多关于序列的细节。现在，你需要知道的是一个序列只是一个有序的项目的集合。

例如 (保存为 for.py):

```
for i in range(1, 5):
    print(i)
else:
    print('for循环结束')
```

输出：

```
$ python for.py
1
2
3
4
for循环结束
```

它是如何工作的：

打印一个数字序列。我们使用内置的 `range` 函数生成这个数字序列。

我们在这里所做的是提供两个数字，`range` 返回一个从第一个数字到第二个数字的一个数字序列。例如，`range(1,5)` 给出序列 `[1, 2, 3, 4]`。默认情况下，`range` 步长值为1。如果我们为 `range` 函数提供第三个参数，那么它这个参数就是新的步长值。例如 `range(1,5,2)` 得到 `[1, 3]`。请记住，数字的范围不超过第二个参数的值，即它不包括第二个数字。

注意，`range()` 生成一个数字序列，在for循环中它一次只生成一个数字。如果你想立刻看到完整的数字序列，使用 `list(range())`。例如，for example，`list(range(5))` 会输出 `[0, 1, 2, 3, 4]`。list(列表)将在数据结构中解释。

`for` 循环然后遍历这个范围，`for i in range(1,5)` 相当于 `for i in [1, 2, 3, 4]`，这就像把序列中的每一个数(或对象)分配给 `i`，一次一个，然后为每个 `i` 值执行该语句块。在本例中，在语句块中我们只是打印它的值。

记住，`else` 部分是可选的。当包括它时，除非遇到break语句，当for循环结束时，它执行一次。

记住，`for..in` 循环可以作用于任何序列。在这里，我们对一个由内建的 `range` 函数生成的一个数字列表，但是一般来说，我们可以使用任何种类对象的任何类型的序列！在后面的章节，我们将详细探讨这个想法。

> C/C++/Java/C#程序要注意 Python的 `for` 循环完全不同于C/C++的 `for` 循环。C#程序员会注意到，在Python中 `for` 循环类似于C#中的 `foreach` 循环。Java程序员会注意到，这也类似于在Java 1.5中的 `for (int i : IntArray)`。
>
> 在C/C++中，如果你想写 `for (int i = 0; i < 5; i++)`，那么在Python中你只要写 `for i in range(0, 5)`。正如您可以看到的，在Python中 `for` 循环更简单，更富有表现力且不易出错。

break语句

`break` 语句是用来跳出一个循环语句，即停止执行一个循环语句，即使循环条件还没有成为 `False` 或序列的项目没有被完全遍历。

很重要的一点是，如果你跳出 `for` 或 `while` 循环，任何相应的循环else块是不执行的。

例子 (保存为break.py):

```
while True:
    s = input('输入一些东西 : ')
    if s == 'quit':
        break
    print('字符串的长度是', len(s))
print('完成')
```

输出：

```
$ python break.py
输入一些东西: Programming is fun
字符串的长度是 18
输入一些东西: When the work is done
字符串的长度是 21
输入一些东西 : if you wanna make your work also fun:
字符串的长度是 37
输入一些东西 : use Python!
字符串的长度是 12
输入一些东西 : quit
完成
```

它是如何工作的：

在这个程序中，我们重复获取用户输入的东西并打印每次输入字符串的长度。我们提供一个特殊的条件--通过检查用户输入是否是 `quit` 来结束程序。我们通过跳出循环停止程序，达到该程序的结束位置。

输入字符串的长度可以使用内建的 `len` 函数。

记住， `break` 语句同样适用于 `for` 循环。

Swaroop 的诗意的Python

在这里输入一个我写的迷你诗：

```
编程自有颜如玉
每当工作完成时
如你也想颜如玉:
    使用Python!
```

continue语句

`continue` 语句是用来告诉Python跳过当前循环块中其余的语句，继续循环的下一次迭代。

例子 (保存为continue.py):

```
while True:
    s = input('输入一些东西: ')
    if s == 'quit':
        break
    if len(s) < 3:
        print('太小')
        continue
    print('输入的东西有足够的长度')
    # 在这做其它各种处理...
```

输出:

```
$ python continue.py
输入一些东西: a
太小
输入一些东西: 12
太小
输入一些东西: abc
输入的东西有足够的长度
输入一些东西: quit
```

它是如何工作的：

在这个程序中，我们接受来自用户的输入，但是只有在输入的字符串至少有3个字符时才处理。因此，我们使用内建的 `len` 函数来获得长度，如果长度小于3，通过使用 `continue` 句，我们跳过块中的其余语句。否则，在循环中的其余语句将被执行--在这做其它各种处理。

注意， `continue` 语句同样适用于 `for` 循环。

小结

我们已经看到了如何使用三个控制流语句-- `if` 、 `while` 和 `for` 以及与它们相关的 `break` 和 `continue` 语句。由于这些都是Python最常用的部分，你需要尽快和他们混得熟一点。

接下来，我们将学习如何创建和使用函数。

继续阅读函数

函数

函数是可重用的程序片段。你可以给一块语句一个名字，并且在你的程序的任何地方使用指定的名字运行任何次数。这就是所谓的函数调用。我们已经使用了许多内置函数，如 `len` 和 `range` 。

在任何有价值软件中，函数的概念都是最重要的基础(对于任何编程语言都一样)，所以，在这一章，我们将探索函数的各各方面。

定义函数使用 `def` 关键字。在这个关键字之后是标识函数的名字，其次是在一对括号中可以附上一些变量名，最后在行的末尾是冒号。接下来是语句块--函数的内容。一个例子将展示这些，这实际上是非常简单的:

例子 (保存为function1.py):

```
def sayHello():
    # 函数的语句块
    print('世界您好!')
# 函数在此结束

sayHello() # 调用函数
sayHello() # 再次调用函数
```

输出：

```
$ python function1.py
世界您好!
世界您好!
```

它是如何工作的：

我们使用上述的语法，定义了一个称为 `sayHello` 函数。这个函数没有参数，因此没有在圆括号中声明变量。函数的参数只是输入到函数中，以便我们可以给它传递不同的值返回相应的结果。

注意，我们可以调用相同的函数两次，这意味着我们不需要再写同样的代码。

函数的参数

一个函数可以带参数--你提供给函数的值，利用这些值该函数可以做一些事情。这些参数就像变量，所不同的是当我们调用函数时这些变量的值已经被定义，当函数运行时，它们已经有了指定的的值。

参数是在函数定义中在一对括号中指定，之间用逗号分隔。当我们调用这个函数，我们以同样的方式提供这些值。注意使用的术语——在函数的定义中给出的名字叫形参，而在函数调用时您提供的值被称为实参。

例子 (保存为func_param.py):

```
def printMax(a, b):
    if a > b:
        print(a, '大')
    elif a == b:
        print(a, '等于', b)
    else:
        print(b, '大')

# 直接给出字面值
printMax(3, 4)

x = 5
y = 7

# 给出参数的变量
printMax(x, y)
```

输出：

```
$ python func_param.py
4 大
7 大
```

它如何工作的：

在这里，我们定义了一个称为 `printMax` 的函数，它拥有 `a` 和 `b` 的两个参数。我们使用简单的 `if..else` 语句找出比较大的数，然后打印较大的数。

我们第一次调用函数 `printMax` ，我们直接提供了数字作为参数。在第二种情况下，我们调用函数使用变量作为参数。 `printMax(x, y)` 把实参 `x` 分配给形参 `a` ， `y` 分配给 `b` 。在这两种情况下， `printMax` 函数以同样方式工作。

局部变量

你在函数定义中声明的变量，他们与在函数外使用的其它同名变量没有任何关系，即变量名称对函数来说是局部的。这叫变量的范围。所有变量都有它们被声明的块的范围，从名称定义的点开始。

例子 (保存为func_local.py):

```
x = 50

def func(x):
    print('x等于', x)
    x = 2
    print('局部变量x改变为', x)

func(x)
print('x一直是', x)
```

输出：

```
$ python func_local.py
x等于50
局部变量x改变为2
x一直是50
```

它是如何工作的：

第一次，我们使用函数体中第一行打印变量*x*的值，Python使用在主块中，函数定义上声明的实参。

接下来，我们给 `x` 赋值为 `2` ，变量为 `x` 对我们的函数来说是局部变量，因此在函数中当我们改变 `x` 的值时，在主块中定义的变量 `x` 不受影响。

最后调用的 `print` 函数，显示在主块中定义的变量 `x` ，说明它不受在前面调用函数的局部变量的影响。

使用全局声明

如果你想给在顶层的程序(即未在任何函数或类之中)定义的变量赋值,那么你必须告诉Python，变量不是局部的，而是全局的。我们使用 `global` 语句，没有 `global` 语句赋值给一个在函数外定义的变量是不可能的。

您可以使用这些在函数外定义的变量的值(假设在函数内没有同名的变量)。然而，这并不鼓励，应该避免，因为这使程序的读者不清楚变量是在哪里定义的，使用 `global` 语句就非常清楚，变量定义在一个最外的块中。

例子 (保存为func_global.py):

```
x = 50

def func():
    global x

    print('x的值是', x)
    x = 2
    print('全局变量x改为', x)

func()
print('x的值是', x)
```

输出：

```
$ python func_global.py
x的值是50
全局变量to改为2
x的值是2
```

它是如何工作的：

`global` 语句用来声明 `x` 是全局变量，当我们在函数内给 `x` 赋值时，它的改变映射到我们在主块中使用的 `x` 的值。

用同样的 `global` 语句可以指定多个全局变量，比如： `global x, y, z` 。

参数默认值

对于一些函数，你可能想要一些参数是可选，即在用户不希望为它们提供值时使用默认值，这在默认的参数值的帮助下完成的。你可以在函数定义中通过在参数名称后使用赋值操作符(`=`)后跟默认值来指定默认的参数值。

注意，默认参数值应该是一个常数。更准确的说，默认的参数值应该是不可变的——这在后面的章节中做了详细解释。现在，只要记住这点。

例子 (保存为 func_default.py):

```
def say(message, times = 1):
    print(message * times)

say('你好')
say('世界', 5)
```

输出：

```
$ python3 func_default.py
你好
世界世界世界世界世界
```

它是如何工作的：

函数 `say` 是用来按照指定的次数打印一个字符串。如果我们不提供一个值，那么在默认情况下字符串只打印一次。为此，我们为参数 `times` 指定一个默认参数值 `1` 。

在第一次使用函数 `say` 时，我们只提供了字符串，它打印字符串一次。在第二次使用 `say` 时，我们提供了字符串和一个实参 `5` 两个参数，说明我们想要 `say` 字符串5次。

> 重要提示 只有在参数列表后面的的参数可以被赋予默认参数值，即在参数列表中，你不能在没有默认值的参数前有有默认参数值的参数。 这是因为，值按位置分配给参数。例如， `def func(a, b=5)` 是有效的，而 `def func(a=5, b)` 是无效的。

参数关键字

如果你有一些有许多参数的函数，您想要指定参数中的一些，那么，你可以通过为参数命名来为它们赋值——这叫做参数关键字——我们使用名称(关键字)而不是位置(我们一直使用的)来指定函数的参数。

这有两个优势，一是，使用函数容易，因为我们不需要担心参数的顺序。二是，如果其他参数有默认参数值，我们可以只给我们想赋值的参数赋值。

例子 (保存为func_key.py):

```
def func(a, b=5, c=10):
    print('a为', a, '和b为', b, '和c为', c)

func(3, 7)
func(25, c=24)
func(c=50, a=100)
```

输出：

```
$ python func_key.py
a为3 和b为7 和c为10
a为25 和b为5 和c为24
a为100 和b为5 和c为50
```

它是如何工作的：

名为 `func` 的函数中有一个参数没有默认参数值，后面两个参数有默认参数值。

第一次使用 `func(3, 7)` ，参数 `a` 得到值 `3` ，参数 `b` 得到值 `7` ，参数 `c` 得到默认值 `10` 。

第二次使用 `func(25, c=24)` , 参数 `a` 按照参数的位置得到值 `25` ，然后参数 `c` 按照参数名，也就是参数关键字，得到值 `24` ，变量 `b` 得到默认值 `5` 。

第三次使用 `func(c=50, a=100)` ，我们为所有给定的值使用了参数关键字。注意，我们为参数 `c` 指定值是在参数 `a` 之前，尽管在函数定义中 `a` 在 `c` 前。

变量参数

有时你可能想定义一个函数，它可以获取参数的任何值，这可以通过使用星号(另存为total.py)实现：

```
def total(a=5, *numbers, **phonebook):
    print('a', a)

    #通过元组遍历全部的参数
    for single_item in numbers:
        print('single_item', single_item)

    #通过字典遍历全部的参数
    for first_part, second_part in phonebook.items():
        print(first_part,second_part)

print(total(10,1,2,3,Jack=1123,John=2231,Inge=1560))
```

输出：

```
$ python function_varargs.py
a 10
single_item 1
single_item 2
single_item 3
Inge 1560
John 2231
Jack 1123
None
```

它是如何工作的：

当我们声明一个星号的参数，如 `*param` ，那么从这一点开始到结束的所有位置的参数都被收集到一个叫param的元组中。

同样,当我们声明一个双星参数，如 `**param` ，那么人那一点开始到结束的所有关键字参数都被收集到一个叫param的字典中。

我们将在数据结构中探讨元组和字典。

return语句

return 语句用来从函数中return(返回)，也就是说跳出函数。同样，我们也可以从函数中选择性地返回一个值。

例子 (保存为func_return.py):

```
def maximum(x, y):
    if x > y:
        return x
    elif x == y:
        return '两个数相等'
    else:
        return y

print(maximum(2, 3))
```

输出：

```
$ python func_return.py
3
```

它是如何工作的：

函数 `maximum` 返回参数中的最大值，在这个例子中是提供给函数的数值。它使用了简单的 `if..else` 语句找到比较大的值，然后返回那个值。

注意，没有一个值的 `return` 语句相当于 `return None` (什么也不返回)。 `None` 是Python中的一个特殊类型，它代表什么也没有。例如，如果一个变量的值是 `None` ，它说明这个变量没有值。

除非你已经写了自己的 `return` 语句，否则，每个函数都默认包含一个 `return None` 语句。通过运行 `print(someFunction())` 你可以看到这一点，这里 `someFunction` 没有使用 `return` 语句，比如：

```
def someFunction():
    pass
```

在Python中pass语句用来说明一个空的语句块。

> 提示：已经有一个叫 `max` 的内建函数能够完成'find maximum'函数的功能 ，因此，只要可能使用这个内建函数。

文档字符串

Python有一个叫文档字符串的好特性，通常用缩写名*docstrings*来指定。 文档字符串是你应该使用的一个重要工具，它对程序文档有助，令其容易理解。令人惊讶的是，当程序实际运行时，我们甚至可以从例如一个函数返回文档字符串。

例子 (保存为func_doc.py):

```
def printMax(x, y):
    '''打印两个数中的最大值。

    两个值必须是整数。'''
    # 如果可能，转换为整数
    x = int(x)
    y = int(y)

    if x > y:
        print(x, '最大')
    else:
        print(y, '最大')

printMax(3, 5)
print(printMax.__doc__)
```

输出：

```
$ python func_doc.py
5 最大
打印两个数中的最大值。

            两个值必须是整数。
```

它是如何工作的：

函数的第一个逻辑行的字符串是那个函数的文档字符串。注意，文档字符串也适用于在后面的章节将要学习的模块和类。

文档的贯例是多行字符串，第一行以大写字母开头以句点(.)结束(注：可以使用中文)，第二行是空行，从第三行开始是详细描述。强烈建议，为你重要的函数写文档字符串要遵循此贯例。

我们可以使用 `printMax` 函数的 `__doc__` (注意，双下划线)属性(属于名字的)访问printMax函数的文档字符串。只要记住，Python把一切事物作为一个对象对待，这也包括函数。我们将在类这一章学习关于对象的更多知识。

如果你在Python中已经使用过 `help()` ，那么你已经看到如何使用文档字符串了!它所做的仅仅是获取函数的 `__doc__` 属性，并以一个整洁的方式显示给你。你可以在上面的函数——在你的程序中仅包括 `help(printMax)` 尝试一下。记得按下 `q` 键，退出 `help` 。

自动化工具可以从你的程序中以这种方式检索文档。因此，我强烈建议，为你写的任何重要函数使用文档字符串。来自Python的自动化工具 `pydoc` 命令使用文档字符串的工作原理类似于 `help()` 。

小结

我们已经见过函数的很多方面，但请注意，我们仍然没有覆盖它们的所有方面。然而，我们已经覆盖了Python函数日常使用的大部分。

接下来，我们将看到如何创建及使用Python模块。

继续阅读模块

模块

您已经看到如何通过一次定义函数在程序中重用代码。如果你想在其它程序中重用一定数量的函数，你将写什么？正如你可能已经猜到了，答案是模块。

编写模块有各种各样的方法，但是最简单的方法是创建一个以.py 为扩展名、包含函数和变量的文件。

编写模块的另一种方式是使用编写Python解释器本身的本机语言，例如，你可以使用C 编程语言编写模块，当它们被编译后，当使用标准的Python解释器时，在你Python代码中可以使用这些模块。

一个模块可以因另一个程序使用其功能而被*imported(导入)*。同样，我们可以使用Python标准库。首先 ,我们将看到如何使用标准库模块。

例子 (保存为 using_sys.py):

```
import sys

print('命令行参数是：')
for i in sys.argv:
    print(i)

print('\n\nPYTHONPATH在', sys.path, '\n')
```

输出:

```
$ python using_sys.py we are arguments
命令行参数是
using_sys.py
we
are
arguments

PYTHONPATH在['/tmp/py',
# 还有很多，这里不一一列出
'/Library/Python/2.7/site-packages',
'/usr/local/lib/python2.7/site-packages']
```

它是如何工作的:

首先,我们使用 `import` 语句*import*导入 `sys` （系统）模块。基本上，这意味着我们我们想告诉Python，我们想使用这个模块。 `sys` 模块包含了与Python解释器和其环境即system系统有关的函数。

当Python执行 `import sys` 语句时，它查找 `sys` 模块。在这里，它是一个内建模块，因此，Python知道到去哪里找到它。

如果它不是一个编译的，也就是用Python写的模块，那么，Python解释器将在 `sys.path` 变量列表中的目录中搜索。如果模块被发现，那么，模块中的代码将运行，对你来说，使用模块变为有效。注意，初始化只有在我们第一次导入一个模块时完成。

在 `sys` 模块中的 `argv` 变量是通过点符号访问的，例如，例如， `sys.argv` 。它清楚地表明，这个名字是 `sys` 模块的一部分。这种方法的另一个优点是，这个名字与你的程序中使用的任何 `argv` 变量都不冲突。

`sys.argv` 变量一个字符串*list*(列表)。具体来说， `sys.argv` 包含命令行参数，也就是使用命令行向你的程序传递参数的列表。

如果您正在使用IDE编写并运行这些程序，在菜单中寻找一种方法来指定命令行参数传递给你的程序。

这里，当我们执行 `python using_sys.py we are arguments` 时，我们使用 `python` 命令和其后的传递给程序的参数运行 `using_sys.py` 模块。Python把命令行参数存储在 `sys.argv` 变量中供我们使用。

记住，运行脚本的名字通常是 `sys.argv` 列表中的第一个参数。因此，在这里将有 `'using_sys.py'` 作为 `sys.argv[0]` ， `'we'` 作为 `sys.argv[1]` ， `'are'` 作为 `sys.argv[2]` 和 `'arguments'` 作为 `sys.argv[3]` 。注意，Python从0而不是1开始数数。

`sys.path` 包含被导入的模块所在的目录名列表。我们发现 `sys.path` 的第一个字符串是空的——这个空字符串表示当前目录是和PYTHONPATH环境变量相同的，同时也是 `sys.path` 变量的一部分。这意味着你可以直接导入位于当前目录中的模块。否则，你将不得不把你的模块存放在 `sys.path` 列表中的一个目录中。

请注意，,当前目录是程序启动的目录。运行 `import os; print(os.getcwd())` 找到你的程序的当前目录。

字节编译的.pyc文件

导入一个模块是一个相对昂贵的事情，所以Python做了一些技巧使它更快。一种方法是创建扩展名为 `.pyc` 的字节编译文件，是Python将程序转换成的一种中间形式。(记得在Python如何工作的简介。当你下次从一个不同程序导入模块时，这种 `.pyc` 文件是有很用的--它将快得多，因为导入模块一部分需要的处理已经完成。同时，这些字节编译的文件是独立于平台的。

注意：这些 `.pyc` 文件通常在与之相应的 `.py` 文件的同一个目录中创建。如果Python在那个目录中没有写入权限，那么 `.pyc` 文件将不会创建。

from ... import语句

如果你想直接导入 `argv` 变量到程序中(为了避免每次为它键入 `sys.`)，那么您可以使用 `from sys import argv` 语句。

> 一般来说，你应该避免使用这个语句，而应该使用 `import` 语句，因为你的程序将避免名称冲突，将更具可读性。

例如：

```
from math import sqrt
print("16的平方根是", sqrt(16))
```

模块的 `__name__`

每个模块都有一个名字，在模块中的语句能够找出它所在的模块的名字。这对于搞清楚模块是否正在运行或被导入这样的特殊用途是很方便的。正如前面提到的，当一个模块被第一次导入时，其所包含的代码被执行。我们可以通过使用这个，根据模块是否被自己使用或从另一个模块被导入，使模块以不同的方式起作用，这些可以通过使用模块的 `__name__` 属性来实现。

例子 (保存为 using_name.py):

```
if __name__ == '__main__':
    print('这个程序正在被自己运行')
else:
    print('我从别的模块被导入')
```

输出：

```
$ python using_name.py
这个程序正在被自己运行
$ python3
>>> import using_name
我从别的模块被导入
>>>
```

它是如何工作的：

每个Python模块有其 `__name__` 定义，如果是 `'__name__'` ，这意味着模块在被用户独立的运行，我们可以采取适当的行动。

制作属于你自己的模块

创建自己的模块是很容易的，你一直在这样做，始终都是！这是因为每个Python程序也是一个模块。 你只需要确保它有一个 `.py` 扩展名。下面的例子会让你明白。

例子 (保存为mymodule.py):

```
def sayhi():
    print('嗨，这是我的模块在讲话。')

__version__ = '0.1'
```

上面的是模块的一个示例。正如您可以看到的，和我们通过的Python程序相比，没有什么特别的。接下来我们要看如何在我们的其它程序中使用这个模块。

记住,该模块要么放置在我们导入它的程序相同的目录中，要么放置在 `sys.path` 目录列表中的一个目录中。

另一个模块(保存为mymodule_demo.py):

```
import mymodule

mymodule.sayhi()
print('版本', mymodule.__version__)
```

输出：

```
$ python mymodule_demo.py
嗨，这是我的模块在讲话。
版本 0.1
```

它是如何工作的：

注意，我们使用相同的点符号来访问模块的成员。Python充分重用相同的符号产生了独特的'Pythonic'的感觉，这样我们不需要不断学习新的方法来做事情。

这是使用 `from..import` 语法的一个版本(保存为mymodule_demo2.py):

```
from mymodule import sayhi, __version__

sayhi()
print('版本', __version__)
```

`mymodule_demo2.py` 和 `mymodule_demo.py` 的输出相同。

注意，如果在导入模块中已经有一个**version**名字的声明，这里会有一个冲突。这也可能是因为它是常见的做法--对于每个模块使用这个名字声明它的版本号。因此，总是推荐选择 `import` 语句，虽然它可能让你的程序有点长。

你还可以使用：

```
from mymodule import *
```

这将导入所有的公共名称如 sayhi，但不会导入**version**，因为它始于双下划线。

> 注意：尽量不要使用 `import *` 这种方式，诸如 `from mymodule import *` 等

> **Python**的禅 Python的一个指导原则是"显式优于隐式"。运行import this去学习更多相关的信息。

`dir` 函数

您可以使用内置的 `dir` 函数列出一个定义对象的标识符。例如,对于一个模块，包括在模块中定义的函数，类和变量。

当你给dir()提供一个模块名字时，它返回在那个模块中定义的名字的列表。当没有为其提供参数时, 它返回当前模块中定义的名字的列表。

例如：

```
$ python

>>> import sys

# 获得属性列表，在这里是sys模块的属性列表
>>> dir(sys)
['__displayhook__', '__doc__', '__excepthook__', '__name__', '__package__', '__s
tderr__', '__stdin__', '__stdout__', '_clear_type_cache', '_compact_freelists',
'_current_frames', '_getframe', 'api_version', 'argv', 'builtin_module_names', '
byteorder', 'call_tracing', 'callstats', 'copyright', 'displayhook', 'dllhandle'
, 'dont_write_bytecode', 'exc_info', 'excepthook', 'exec_prefix', 'executable',
'exit', 'flags', 'float_info', 'getcheckinterval', 'getdefaultencoding', 'getfil
esystemencoding', 'getprofile', 'getrecursionlimit', 'getrefcount', 'getsizeof',
'gettrace', 'getwindowsversion', 'hexversion', 'intern', 'maxsize', 'maxunicode
', 'meta_path', 'modules', 'path', 'path_hooks', 'path_importer_cache', 'platfor
m', 'prefix', 'ps1', 'ps2', 'setcheckinterval', 'setprofile', 'setrecursionlimit
', 'settrace', 'stderr', 'stdin', 'stdout', 'subversion', 'version', 'version_in
fo', 'warnoptions', 'winver']

# 获得当前模块的属性列表
>>> dir()
['__builtins__', '__doc__', '__name__', '__package__', 'sys']

# 创建了一个新变量 'a'
>>> a = 5

>>> dir()
['__builtins__', '__doc__', '__name__', '__package__', 'a', 'sys']

# 删除/移除一个名字
>>> del a

>>> dir()
['__builtins__', '__doc__', '__name__', '__package__', 'sys']

>>>
```

它是如何工作的：

首先，我们看到在导入 `sys` 模块上使用使用 `dir` 。我们能看到模块包含的巨大的属性列表。

然后，我们使用没有传递参数的 `dir` 函数。默认情况下，它返回模块的属性的列表。注意，导入模块的列表仍然是这个列表的一部分。

为了看到 `dir` 在起作用，我们定义了一个新的变量，并为其赋值，然后检查 `dir` ，我们发现列表中添加了一个同名变量。我们使用 `del` 语句移除当前模块的变量或属性，在 `del` 函数的输出中变化再次得到体现。

关于 `del` 的一点注意事项--这个语句用于删除一个变量/属性，语句运行后，这里是 `del a` ，你不能再访问变量 `a` --就像它从来根本没有存在过。

注意，`dir()` 函数对任何对象都起作用。例如，运行 `dir('str')` 来学习 `str` (string)类型的更多知识。

打包（Packages）

现在，你必须开始观察组织你的程序的层次结构。变量通常在函数内部。函数和全局变量通常在模块内部。如果你想组织模块？这就到了牵涉到打包的地方了。

包只是模块的文件夹，使用一个特殊的 `__init__.py` 文件，指示Python，这个文件夹是特殊的，因为它包含Python模块。

假设你想创建一个叫做'世界'的程序包，分装'亚洲'、'非洲'等等，分包按序包含'印度'、'马达加斯加'等等。

这是你的文件结构：

```
- <在sys.path中现有的一些文件夹>/
    - world/
        - __init__.py
        - asia/
            - __init__.py
            - india/
                - __init__.py
                - foo.py
        - africa/
            - __init__.py
            - madagascar/
                - __init__.py
                - bar.py
```

包只是为了分层次组织模块的方便。在标准库中，你会看到包的许多实例。

小结

就像函数是可重用的部分的程序一样，模块也是可重用的程序。包是组织模块的另一个层次结构。来自Python的标准库，是包和模块的集合中的一个例子。

我们已经看到了如何使用这些模块和创建我们自己的模块。

接下来，我们将学习一些有趣的称为数据结构的概念。

继续阅读数据结构

数据结构

数据结构用来将一些数据组织在一起。换句话说，它们是用来存储一系列相关的数据。

在Python中有四种内建数据结构--列表、元组、字典和集合，我们将看到如何使用它们中的每一个，它们是怎样使我们的生活更容易的。

列表

`列表` 是一种数据结构，它保存条目的有序集合。例如，你可以在列表中存储一个序列。这很容易想象，你想像一下购物清单,那里有你要购买物品的一个清单。除非在你的清单上每一行列有一个单独物品，然而，在python中，你在它们中间放上逗号。

条目的列表应包含在方括号内，以便Python明白你在指定一个列表。一旦您创建了一个列表，你可以添加、删除或是搜索列表中的条目。因为我们可以添加和删除条目，我们说一个列表是一个可变的数据类型，即这种类型可以更改。

对象和类的快速介绍

尽管直到现在，我一直推迟讨论对象和类，现在需要一个小小的解释，这样你可以更好的理解列表。我们将在后面章节中详细探讨这一课题。

列表是使用对象和类的一个例子。当我们使用一个变量 i，为它分配一个值，例如把整数5赋值给它，你可以认为它是创建一个类为int（即类型）的对象(即实例)i。事实上，你可以阅读 `help(int)` 更好地理解这一点。

类也有方法，也就是为了使用而定义的只关于那个类的函数，只有当你有那个类的对象时，你才可以使用这些函数.例如，Python为 `list` （列表）类提供了一个 `append` 方法，它允许你在列表的整改添加一个条目。例如， `mylist.append('an item')` 将给列表 `mylist` 添加字符串。注意，我们使用点操作符访问对象的方法。

类也有字段，除了为了使用而定义的只与那个类相关的变量，它什么也没有。只有当你有那个类的对象时，你可以使用那些变量或名字。字段孔明通过点操作符访问的。例如， `mylist.field` 。

例子 (保存为using_list.py):

```
# 这是我的购物清单
shoplist = ['苹果', '芒果', '胡萝卜', '香蕉']

print('我要买', len(shoplist), '个物品。')

print('清单是：', end=' ')
for item in shoplist:
    print(item, end=' ')

print('\n我还要买大米。')
shoplist.append('大米')
print('现在我的清单是', shoplist)

print('现在我将要为我的清单排序')
shoplist.sort()
print('排序后的购物清单是', shoplist)

print('我要买的第一个物品是', shoplist[0])
olditem = shoplist[0]
del shoplist[0]
print('我已经买了', olditem)
print('现在我的清单是', shoplist)
```

输出：

```
$ python using_list.py
我要买 4 个物品。
清单是: 苹果 芒果 胡萝卜 香蕉
我还要买大米。
现在我的清单是 ['苹果', '芒果', '胡萝卜', '香蕉', '大米']
现在我将要为我的清单排序
排序后的购物清单是 ['大米', '胡萝卜', '芒果', '苹果', '香蕉']
我要买的第一个物品是 大米
我已经买了大米
现在我的清单是 ['胡萝卜', '芒果', '苹果', '香蕉']
```

它是如何工作的:

变量 `shoplist` 是将要去超市的人的一个购物清单。在 `shoplist` 中，我们只存储了要买的物品的名字的字符串，你可以向清单中添加包括数字甚至是其它清单的任何对象。

我们也使用了循环 `for..in` 遍历清单中的所有条目。到现在为止，你必须认识到一个列表也是一个序列。序列的特性将在后面的章节中讨论。

注意，在 `print` 函数中使用 `end` 关键字参数，表明输出以一个空格结束而不是通常的换行。

接下来，和前面讨论过的一样，我们使用列表对象的 `append` 方法向列表中添加一个项目。然后，我们只把列表简单地传递给 `print` 语句，整洁地打印列表的内容，以检查这个条目确实添加到了列表中。

然后，我们使用列表对象的 `sort` 方法为列表排序。 这个方法作用到列表本身，并不返回一个修改过的列表，理解这一点很很重要，它不同于对字符串的操作。这也是为什么我们列表是可修改的，而字符串是不可修改的原因。

然后，我们在超市购买了一个物品，我们想把它从购物清单中移除，通过使用 `del` 语句来实现。这里，我们提到我们想要移除清单中的哪个物品，del语句为我们将它从清单中移除。我们指定，我们想从清单移除第一项，因此，我们使用 `del shoplist[0]` (记住，Python从0开始计数）。

如果你想知道列表对象定义的所有方法，详见 `help(list)` 。

元组

元组是用来容纳多个对象。认为它们是类似于列表，但是没有列表给你的广泛功能。元组的一个主要特征是他们是不可变，像字符串，即您不能修改元组。

元组是通过在一对可选的圆括号中，项目之间用逗号分隔来定义的。

元组通常用在，一个语句或一个用户定义的函数能够安全地假设为值的集合，即值的元组，不会改变。

例子 (保存为using_tuple.py):

```
# 我推荐使用括号表示元组的开始和结束，尽管括号是可选的。
# 毕竟显式声明比隐式声明更加直观
zoo = ('蟒蛇', '大象', '企鹅') # 记住圆括号是可选的
print('动物园中动物有数量有', len(zoo))

new_zoo = '猴子', '骆驼', zoo
print('在新动物园中笼子的数量是', len(new_zoo))
print('在新动物园所有的动物是', new_zoo)
print('从老动物园中带来的动物是', new_zoo[2])
print('从老动物园带来最后的动物是', new_zoo[2][2])
print('在新动物园中动物的数量有', len(new_zoo)-1+len(new_zoo[2]))
```

输出：

```
$ python using_tuple.py
动物园中动物有数量有 3
在新动物园中笼子的数量是 3
在新动物园所有的动物是 ('猴子', '骆驼', ('蟒蛇', '大象', '企鹅'))
从老动物园中带来的动物是 ('蟒蛇', '大象', '企鹅')
从老动物园带来最后的动物是 企鹅
在新动物园中动物的数量有 5
```

它是如何工作的：

变量 `zoo` 指的是一个物品的元组。我们看到 `len` 函数可以用来获取元组的长度。这也表明，一个元组同样也是一个序列。

因为老动物园zoo将要关闭，我们现在将这些动物迁移到一个新的动物园 `new_zoo` 。因此，（新动物园）new_zoo的tuple包含一些已经存在的动物以及从老动物园zoo带来的动物。回到现实，请注意，在一个元组中的元组不失去其特性。

就像列表一样，我们可以通过在一对方括号中指定条目的位置，访问元组中的物品。这被称为索引操作符。我们通过指定 `new_zoo[2]` 访问新动物园 `new_zoo` 中的第三项，通过指定 `new_zoo[2][2]` 访问新动物园 `new_zoo` 的第三项。一旦理解这个习语，这是非常简单的。

> **有0个或1个条目的元组**
>
> 一个空的元组由一对空的括号如 `myempty = ()` 组成。 然而，只有一个对象的元组并非如此简单。你必须通过在第一个对象（唯一的一个）后紧跟一个逗号来指定它，这样Python可以区分是一个元组还是一个表达式中一个对象的括号，例如，如果你想定义一个只含一个对象这为 `2` 的元组，你必须使用 `singleton = (2 , )` 。

> **Perl程序员应该注意**
>
> 在一个列表中的列表不会失去其特性，也就是说并不像在Perl中夷为平地。这同样适用于在一个元组中的一个元组，或在一个列表中的元组，或在一个元组中的列表等。就Python而言，他们只是存储在另一个对象中的一个对象。

字典

字典就像一个地址簿，在那里你只通过知道他/她的名字，就可以找到地址或联系人详细信息。也就是说，我们使用键(姓名)与值(细节)相联系。注意，键必须是独一无二的，就像如果有两个完全相同的名字的人，你无法找到正确的信息。

注意，字典的关键字你只能使用不可变的对象(比如字符串)，你可以使用不可变或可变的对象作为字典的值。这基本上意味着，简单地说，对于键你只能使用简单对象。

在字典中的一对键和值是通过使用冒号指定的，如， `d = {key1 : value1, key2 : value2 }` 。注意，键值对用冒号分隔，彼此之间以逗号分隔，所有这些都是包含在一对大括号中。

记住，在字典中键-值对不以任何方式排序。如果你想要一个特定的顺序，那么你将不得不在使用前自己排序。

你将要使用的字典是 `dict` 类的对象或实例。

例子 (保存为 using_dict.py):

```
# 'ab'是英文address book(地址簿)的首个字母

ab = {  'Swaroop'   : 'swaroop@swaroopch.com',
        'Larry'     : 'larry@wall.org',
        'Matsumoto' : 'matz@ruby-lang.org',
        'Spammer'   : 'spammer@hotmail.com'
    }

print("Swaroop的地址是", ab['Swaroop'])

# 删除一个键-值对
del ab['Spammer']

print('\n地址薄中有 {0} 个联系人\n'.format(len(ab)))

for name, address in ab.items():
    print('联系人 {0} 的地址是 {1}'.format(name, address))

# 添加一个键-值对
ab['Guido'] = 'guido@python.org'

if 'Guido' in ab:
    print("\nGuido的地址是", ab['Guido'])
```

输出：

```
$ python using_dict.py
Swaroop的地址是 swaroop@swaroopch.com

地址薄中有 3 个联系人

联系人 Larry 的地址是 larry@wall.org
联系人 Matsumoto 的地址是 matz@ruby-lang.org
联系人 Swaroop 的地址是 swaroop@swaroopch.com

Guido的地址是 guido@python.org
```

它是如何工作的：

我们使用已经讨论过的符号创建字典 `ab` 。然后我们通过使用在列表和元组中讨论过的索引操作符——指定关键字来访问键-值对，遵守简单的语法。

我们可以使用我们的老朋友—— `del` 语句删除键值对，我们简单地指定字典和要删除的关键字的索引操作符，并将它传递给 `del` 语句。对于这个操作，没有必要知道对应于关键字的值。

接下来，我们我们使用字典的 `items` 方法，访问字典的每个键-值对的。它返回一个元组的列表，每个元组包含一对值--关键字及紧随其后的值。我们检索这对值并使用for..in循环为每一对分配给相应的变量 `name` 和 `address` ，然后在 `for..in` 块中打印这些值。

我们可以通过简单地使用索引操作符来访问一个键并分配值的方式添加新的键值对，像上面的例子中我们所做的添加Guido。

我们可以使用 `in` 操作符来检查一个键值对是否存在。

字典 `dict` 类的列表方法，请看 `help(dict)` 。

> 关键字参数和字典
>
> 如果你已经在函数中使用过了关键字参数，那么你已经接触过字典了。想象一下，这个键值对是在函数定义的参数列表中指定的，而当你在函数中访问变量，它只是访问字典的一个键(在编译器设计术语中称为符号表)。

序列

列表、元组和字符串都序列的一个实例，但是什么是序列，它们为什么如此特殊呢 ？

主要特点是成员测试，即in(在)和not in(不在)表达式中和索引操作，这使我们在一个序列中能够直接获取一个特定的对象。

上面提到的——列表、元组和字符串这三种类型的序列，也有允许我们找回一彼序列即序列的一部分的切片操作。

例子 (保存为seq.py):

```
shoplist = ['苹果', '芒果', '胡萝卜', '香蕉']
name = 'swaroop'

# Indexing or 'Subscription' operation
print('第0项是', shoplist[0])
print('第1项是', shoplist[1])
print('第2项是', shoplist[2])
print('第3项是', shoplist[3])
print('第-1项是', shoplist[-1])
print('第-2项是', shoplist[-2])
print('第0个字符是', name[0])

# 一个列表的切片
print('第1项到第3项是', shoplist[1:3])
print('第2项到末尾是', shoplist[2:])
print('第1到-1项是', shoplist[1:-1])
print('开头到结尾是', shoplist[:])

# 字符串的切片
print('第1到第3个字符是', name[1:3])
print('第2到末尾的字符是', name[2:])
print('第1到-1的字符是', name[1:-1])
print('从头到尾的字符是', name[:])
```

输出：

```
$ python seq.py
第0项是 苹果
第1项是 芒果
第2项是 胡萝卜
第3项是 香蕉
第-1项是 香蕉
第-2项是 胡萝卜
第0个字符是 s
第1项到第3项是 ['芒果', '胡萝卜']
第2项到末尾是 ['胡萝卜', '香蕉']
第1到-1项是 ['芒果', '胡萝卜']
开头到结尾是 ['苹果', '芒果', '胡萝卜', '香蕉']
第1到第3个字符是 wa
第2到末尾的字符是 aroop
第1到-1的字符是 waroo
从头到尾的字符是 swaroop
```

它是如何工作的：

首先，我们看看如何使用索引来获得一个序列的个别项，这也称为订阅操作。当你在方括号中指定一个数字对应一个序列中的某项，如上所示，Python会为你取得序列中相对应位置的项。记住，Python从0开始数数。因此，在序列 `shoplist` 中， `shoplist[0]` 取第一项和 `shoplist[3]` 获取第四项。

索引也可以是负数，在这种情况下，这个位置从序列的结尾开始计算。因此，`shoplist[-1]` 指的是序列的最后一项， `shoplist[-2]` 取倒数第二个项。

这个切片操作是通过指定序列的名称后面加上一个方括号，方括号中有一对可选的用冒号分隔的数。注意，这非常类似于你到现在一直使用的索引操作，记住这些数字是可选的但冒号不是。

在切片操作中的第一个数字(在冒号前)是切片开始的位置，第二个数字(在冒号后)是切片停止的位置。如果第一个数字没有指定，Python会从序列开头开始，如果第二个数字被冷落，Python会在序列的末尾停止。注意，返回的切片在开始位置开始，在结束位置前结束，也就是说，返回的切片包含开始位置，但不包含结束位置。

因此， `shoplist[1:3]` 返回序列的切片从位置1开始，包括位置2，但是在位置3停止，因此，返回两个项目的切片。同样， `shoplist[:]` 返回整个序列的一个副本。

你也可以使用负位置做切片。负数用于从序列的结尾开始。例如， `shoplist[:-1]` 将返回一个不包括序列最后一项，但包含了其它一切的切片。

你也可以为切片提供第三个参数，这是切片的步长(默认情况下，步长为1)：

```
>>> shoplist = ['苹果', '芒果', '胡萝卜', '香蕉']
>>> shoplist[::1]
['苹果', '芒果', '胡萝卜', '香蕉']
>>> shoplist[::2]
['苹果', '胡萝卜']
>>> shoplist[::3]
['苹果', '香蕉']
>>> shoplist[::-1]
['香蕉', '胡萝卜', '芒果', '苹果']
```

注意，当步长是2时，我们获得位置0、2、......的项目，当步长是3晨，我们获得位置是0、3、等等的项目。

使用Python解释器的交互式提示，尝试指定切片的不同组合，以便你可以立刻看到结果。序列的一大好处是，你可以以同样的方式访问元组、列表和字符串！

集合

集合是简单对象的无序集合，用于一个集合中对象的存在比它的顺序或发生多少次更重要的时候。

使用集合，你可以测试成员，它是否是集合的子集以及找到两个集合的交集，等等。

```
>>> bri = set(['巴西', '俄罗斯', '印度'])
>>> '印度' in bri
True
>>> 'usa' in bri
False
>>> bric = bri.copy()
>>> bric.add('中国')
>>> bric.issuperset(bri)
True
>>> bri.remove('俄罗斯')
>>> bri & bric # 或者 bri.intersection(bric)
{'巴西', '印度'}
```

它是如何工作的：

这个例子是非常一目了然的，因为它涉及到学校教的数学的基本集合理论。

关联

当你创建一个对象，并赋给它一个值，该变量只是指向对象，并不代表对象本身！也就是说，变量名称指向你电脑中内存中的存储对象的那部分。这就是所谓的把名字绑定给对象。

一般来说，你不需要担心这个，但是对引用有一个需要你注意的微妙的影响。

例子 (保存为reference.py):

```
print('简单的分配')
shoplist = ['苹果', '芒果', '胡萝卜', '香蕉']
# mylist是指向同一对象的另一个名字！
mylist = shoplist

# 我买到了第一项物品，因此我从清单中移除它
del shoplist[0]

print('shoplist是', shoplist)
print('mylist是', mylist)
# 注意shoplist和mylist都打印没有'苹果'的相同的清单
# 证明它们指向相同的对象

print('通过制作完整的切片复制')
# 通过制作完整的切片复制
mylist = shoplist[:]
# 移除第一项
del mylist[0]

print('shoplist是', shoplist)
print('mylist是', mylist)
# 注意，现在两个清单不同
```

输出：

```
$ python reference.py
简单的分配
shoplist是 ['芒果', '胡萝卜', '香蕉']
mylist是 ['芒果', '胡萝卜', '香蕉']
通过制作完整的切片复制
shoplist是 ['芒果', '胡萝卜', '香蕉']
mylist是 ['胡萝卜', '香蕉']
```

它是如何工作的:

在注释中有更多有用的解释。

记住,如果你想要复制一个列表或这种类型的序列或复杂的对象(而不是简单的对象如整数)，那么您必须使用切片操作复制。如果你只是用另一个变量名指定，两个变量将“关联”到相同的对象，如果你不小心，这可能会引起麻烦。

> **Per程序员需要注意**
>
> 记住，列表的一个赋值语句并不创建一个副本。你必须使用切片操作复制序列。

关于字符串的更多

之前，我们已经详细讨论了字符串。在这能了解更多吗？嗯，你知道吗，字符串也是对象和也有做任何事情的方法——从检查的部分字符串到从字符串中分离。

在程序中你使用的字符串都是str类的对象，在下面的例子中将演示这个类的一些有用的方法，这些方法的完整列表，请看help(str)。

例子 (保存为 str_methods.py):

```python
# 这是一个字符串对象
name = 'Swaroop'

if name.startswith('Swa'):
    print('是的，字符串以"Swa"开始')

if 'a' in name:
    print('是的，它包含字符串"a"')

if name.find('war') != -1:
    print('是的，它包含字符串"war"')

delimiter = '_*_'
mylist = ['巴西', '俄罗斯', '印度', '中国']
print(delimiter.join(mylist))
```

输出:

```
$ python str_methods.py
是的，字符串以"Swa"开始
是的，它包含字符串"a"
是的，它包含字符串"war"
巴西_*_俄罗斯_*_印度_*_中国
```

它是如何工作的：

在这里，我们看到字符串的很多方法在起作用。`startswith` 方法是用来找出字符串是否以给定的字符串开始的。`in` 操作符是用来检查一个给定的字符串是否是一个字符串的一部分。

`find` 方法用于定位给定的子字符串在字符串内的位置，如果不能成功找到子字符串它返回-1。str类也有一个整洁的方法来 `join` (连接)一个序列的字符串，用充当分隔符的字符串连接序列中每个条目，返回一个由它生成的巨大的字符串。

小结

我们详细探索了Python各种内建的数据结构，写合理大小的程序，这些数据结构是至关重要的。

现在，我们有很多Python的基本知识已经到位，下面，我们看看如何设计和写一个真实的Python程序。

继续阅读解决问题

解决问题

我们已经探索过了Python语言的各种部分，现在我们通过设计和编写一个做有用事情的程序，看一看如何将所有这些组合在一起，学习如何自己编写一个Python脚本可以实现这个想法。

问题

我们想要解决的问题是：

> 我需要一个为我所有重要的程序创建备份的一个程序。

尽管这是一个简单的问题，但是我们没有着手解决这个问题的足够的信息。多一点的分析是必需的，例如，我们如何指定哪一个文件需要备份？他们是怎样存储的？

在得当的问题分析后，我们设计我们的程序。我们为程序如何工作列一个列表，在本例中，我创建了我希望它如何工作的以下列表。如果你做这个设计，你可能不会拿出同样的分析，因为每个人都有自己做事的方式，这是非常好的。

- 在列表中指出需要备份的文件和目录。
- 备份必须存储在一个主备份目录中。
- 备份的文件压缩到一个压缩文件中。
- 压缩文件的名称是当前的日期和时间。
- 在标准的 Linux/Unix 发行版上，我们默认使用标准的zip命令。注意，只要它有一个命令行，你可以使用任何你想要归档的命令。

，Windows用户可以从GnuWin32项目页 安装，并向你的系统环境变量 PATH追加 C:\Program Files\GnuWin32\bin，这和为识别Python命令我们所做的类似

> **Windows**用户
>
> Windows用户可以安装在GnuWin32项目主页安装 `zip` 命令并且将 `C:\Program Files\GnuWin32\bin` 添加到你的环境变量 `PATH` 中，就好像我们配置python命令行一样。

解决方案

由于我们的程序的设计现在相当稳定，我们可以写实现解决方案的代码。

保存为backup_ver1.py:

```
import os
import time

# 1. 在列表中指出需要备份的文件和目录。
# 在Windows中的例子:
# source = ['"C:\\My Documents"', 'C:\\Code']
# 在Mac OS X和Linux中的列子:
source = ['"C:\\My Documents"', 'C:\\Code']
# 注意我们在有空格的名字的字符串内不得不使用双引号。

# 2. 备份必须存储在一个主备份目录中。
# 在Windows中的例子:
# target_dir = 'E:\\Backup'
# 在Mac OS X和Linux中的例子:
target_dir = 'E:\\Backup'
# 记住把它改为你要使用的目录

# 3. 备份的文件压缩到一个压缩文件中。
# 4. 压缩文件的名称是当前的日期和时间。
target = target_dir + os.sep + \
         time.strftime('%Y%m%d%H%M%S') + '.zip'

# 如果目录不存在就创建
if not os.path.exists(target_dir):
    os.mkdir(target_dir)  # 创建目录

# 5. 我们使用zip命令把文件压缩到一个压缩文件中
zip_command = "zip -qr {0} {1}".format(target,
                                       ' '.join(source))

# 运行备份
print("Zip命令为:")
print(zip_command)
print("运行:")
if os.system(zip_command) == 0:
    print('成功备份到', target)
else:
    print('备份失败')
```

输出:

```
$ python backup_ver1.py
Zip命令为:
zip -r /Users/swa/backup/20140328084844.zip /Users/swa/notes
运行:
  adding: Users/swa/notes/ (stored 0%)
  adding: Users/swa/notes/blah1.txt (stored 0%)
  adding: Users/swa/notes/blah2.txt (stored 0%)
  adding: Users/swa/notes/blah3.txt (stored 0%)
成功备份到 E:\Backup\20080702185040.zip
```

现在，我们是在测试我们的程序能否正常工作的测试阶段。如果它不像预期的那样,则我们必须调试我们的程序，也就是从程序中去掉*bug*(错误)。

如果上面的程序不为你工作，将打印出来的 `Zip命令为:` 下面那一行拷贝一下，然后在shell(GNU/Linux和Mac OS X中)/ `cmd` (Windows中)里面粘贴，看看有哪些报错，然后尝试修复它。如果这个命令失败,检查压缩命令手册，是什么可能是错的。如果这个命令成功，然后检查Python程序是否和上面的程序完全匹配。

它是如何工作的：

你会注意到，我们是如何以一个循序渐进的方式将我们的设计转换成代码的。

我们通过先导入和使用 `os` 和 `time` 模块，然后，我们在 `source` 清单中指定要备份的文件和目录，目标目录存储在变量 `target_dir` 中，这是我们要存储的所有的备份的文件的地方，我们将要创建的压缩文件的名称是使用 `time.strftime()` 函数由当前日期和时间生成的。它也包含 `.zip` 扩展名，将存储在 `target_dir` 目录中。

注意，变量 `os.sep` 的使用--目录的分隔符依你的操作系统而定，在GNU/Linux和Unix中是 `'/'` ，在Windows中是 `'\\'` ，在Mac OS中是 `':'` 。使用 `os.sep` 而不是直接使用这些字符将使我们的程序更具可移植性，在所有这些系统上都能工作。

`time.strftime()` 函数获取一个技术参数，像在上面的程序中我们已经使用过的。 `%Y` 参数将被替换为带着世纪的年份数字。 `%m` 参数将被一个位于 `01` 到 `12` 的数字替换，如此等等。这种参数的完整列表可Python参考手册中找到。

我们创建目标文件的名称zip文件使用了加法操作符连接字符串，它将两个字符串连接在了一起，并返回一个新的字符串。然后,我们创建一个包含我们要执行的命令的字符串 `zip_command` 。你可以通过在shell(GNU/Linux终端或DOS提示符)运行来检查这个命令的工作。

我们使用的 `zip` 命令有一些选项和参数。 `-q` 选项用于表明zip命令应该**quietly(**默默地**)**工作， `-r` 选项指定zip命令应该**recursively(**递归地**)**工作，即它应该包括所有的子目录和文件。这两个选项组合在一起可缩写为 `-qr` 。要创建的压缩文件名后的选项后面紧跟要备份的文件和目录列表，我们使用字符串的 `join` 方法，这种方法我们已经知道如何使用，将 `source` 列表转换成一个字符串。

然后，我们终于使用 `os.system` 函数运行命令。os.system函数运行命令就仿佛在系统上也就是shell上运行它，如果命令运行成功，它返回 `0` ，否则返回一个错误号。

根据命令的结果，我们打印相应的消息，备份失败或成功。

就是这样，我们已经创建了一个备份我们重要文件的一个脚本!

> **Windows**用户应注意
>
> 您还可以使用原始字符串，而不是双反斜杠转义序列，例如，使用`'C:\\Documents'`或`r'C:\Documents'`。然而，不要使用`'C:\Documents'`，因为你最终用一个未知的转义序列`\D`。

现在，我们有一个能够工作的备份脚本，当我们想要备份文件时，我们可以用它。这被称为软件的操作阶段或部署阶段。

上面的程序正常工作，但(通常)第一个程序不会像你期望的那么样工作。例如，如果没有正确地设计程序或当键入代码时如果你犯了一个错误等，可能出现问题。相应地，你将不得不回到设计阶段或你需要调试您的程序。

第二版

第一个版本的脚本工作了。然而，我们还可以做一些改进，以便每天工作得更好。这被称为软件的维护阶段。

我觉得有用的改进之一是有一个更好的文件命名机制——在一个目录中，使用时间作为文件的名称，使用当前日期作为主备份目录中的一个目录。第一个优势是，你的备份以分层以方式存储，因此它更容易管理。第二个优势是，文件名短得多。第三个优势是单独的目录将帮助你检查每天是否创建了一个备份，如果某一天你备份了，将会创建一个目录。

保存为 backup_ver2.py:

```
import os
import time

# 1. 在列表中指出需要备份的文件和目录。
# 在Windows中的例子:
# source = ['"C:\\My Documents"', 'C:\\Code']
# 在Mac OS X和Linux中的列子:
source = ['"C:\\My Documents"', 'C:\\Code']
# 注意我们在有空格的名字的字符串内不得不使用双引号。

# 2. 备份必须存储在一个主备份目录中。
# 在Windows中的例子:
# target_dir = 'E:\\Backup'
# 在Mac OS X和Linux中的例子:
target_dir = 'E:\\Backup'
# 记住把它改为你要使用的目录

# 如果目录不存在就创建
if not os.path.exists(target_dir):
    os.mkdir(target_dir)  # 创建目录

# 3. 备份的文件压缩到一个压缩文件中。
# 4. 在主备份目录中创建一个子目录，名字是当前的日期。
today = target_dir + os.sep + time.strftime('%Y%m%d')
# zip文件的名字是当前的时间。
now = time.strftime('%H%M%S')

# zip文件的完整路径
target = today + os.sep + now + '.zip'

# 如果子目录不存在就创建它
if not os.path.exists(today):
    os.mkdir(today)
    print('成功创建子目录：', today)

# 5. 我们使用zip命令把文件压缩到一个压缩文件中
zip_command = "zip -qr {0} {1}".format(target,
                                       ' '.join(source))

# 运行备份
print("Zip命令为:")
print(zip_command)
print("运行:")
if os.system(zip_command) == 0:
    print('成功备份到', target)
else:
    print('备份失败')
```

输出：

```
$ python backup_ver2.py
成功创建子目录：/Users/swa/backup/20140329
Zip命令为:
zip -r /Users/swa/backup/20140328084844.zip /Users/swa/notes
运行:
  adding: Users/swa/notes/ (stored 0%)
  adding: Users/swa/notes/blah1.txt (stored 0%)
  adding: Users/swa/notes/blah2.txt (stored 0%)
  adding: Users/swa/notes/blah3.txt (stored 0%)
成功备份到 E:\Backup\20080702185040.zip
```

它是如何工作的:

大部分程序还保留了原样，变化是，我们使用 `os.path.exists` 函数检查在主备份目录中是否存在以当前日期为名字的目录，如果不存在，我们使用 `os.mkdir` 函数创建它。

第三版

当我们做一些备份时，第二版工作起来很好了。但当有许多备份时，我发现区分为什么备份是很困难的。例如，对一个程序或描述做一些重要的改变，然后我想知道这些变化与压缩文件的名字有什么联系。这个可以通过为压缩文件的名字附加上一个用户提供的注释而轻易实现。

注意：下面的程序不工作，所以不要惊慌，请继续，因为在这里有一个教训。 保存为 backup_ver3.py:

```python
import os
import time

# 1. 在列表中指出需要备份的文件和目录。
# 在Windows中的例子:
# source = ['"C:\\My Documents"', 'C:\\Code']
# 在Mac OS X和Linux中的列子:
source = ['"C:\\My Documents"', 'C:\\Code']
# 注意我们在有空格的名字的字符串内不得不使用双引号。

# 2. 备份必须存储在一个主备份目录中。
# 在Windows中的例子:
# target_dir = 'E:\\Backup'
# 在Mac OS X和Linux中的例子:
target_dir = 'E:\\Backup'
# 记住把它改为你要使用的目录

# 如果目录不存在就创建
if not os.path.exists(target_dir):
    os.mkdir(target_dir)  # 创建目录
```

```
# 3. 备份的文件压缩到一个压缩文件中。
# 4. 在主备份目录中创建一个子目录，名字是当前的日期。
today = target_dir + os.sep + time.strftime('%Y%m%d')
# zip文件的名字是当前的时间。
now = time.strftime('%H%M%S')

# 提示用户输入zip文件名中附加的注释
comment = input('输入注释 --> ')
# Check if a comment was entered
if len(comment) == 0:
    target = today + os.sep + now + '.zip'
else:
    target = today + os.sep + now + '_' +
        comment.replace(' ', '_') + '.zip'

# 如果子目录不存在就创建它
if not os.path.exists(today):
    os.mkdir(today)
    print('成功创建子目录：', today)

# 5. 我们使用zip命令把文件压缩到一个压缩文件中
zip_command = "zip -qr {0} {1}".format(target,
                                       ' '.join(source))

# 运行备份
print("Zip命令为:")
print(zip_command)
print("运行:")
if os.system(zip_command) == 0:
    print('成功备份到', target)
else:
    print('备份失败')
```

输出：

```
$ python backup_ver3.py
  File "backup_ver3.py", line 25
    target = today + os.sep + now + '_' +
                                        ^
SyntaxError: invalid syntax
```

这怎么（不）工作：

这个程序不工作！Python说有语法错误这意味着脚本不满足Python预计的结构。当我们观察Python给出的错误，它还告诉我们它检测到错误的地方。所以我们从那一行开始调试我们的程序。

在仔细观察后，我们发现单一的逻辑行被分成两个物理行，但我们没有指定这两个物理行属于同一个逻辑行。基本上，Python发现在那个逻辑行添加操作符(+)没有任何操作对象，因此不知道如何继续。记住，我们可以通过在物理行的结束位置使用反斜杠指定当前行与下一物理行是连续的。所以，我们要改正我们的程序。我们找到错误时的这样修正叫做修复bug。

第四版

保存为 backup_ver4.py:

```
import os
import time

# 1. 在列表中指出需要备份的文件和目录。
# 在Windows中的例子:
# source = ['"C:\\My Documents"', 'C:\\Code']
# 在Mac OS X和Linux中的列子:
source = ['"C:\\My Documents"', 'C:\\Code']
# 注意我们在有空格的名字的字符串内不得不使用双引号。

# 2. 备份必须存储在一个主备份目录中。
# 在Windows中的例子:
# target_dir = 'E:\\Backup'
# 在Mac OS X和Linux中的例子:
target_dir = 'E:\\Backup'
# 记住把它改为你要使用的目录

# 如果目录不存在就创建
if not os.path.exists(target_dir):
    os.mkdir(target_dir)  # 创建目录

# 3. 备份的文件压缩到一个压缩文件中。
# 4. 在主备份目录中创建一个子目录，名字是当前的日期。
today = target_dir + os.sep + time.strftime('%Y%m%d')
# zip文件的名字是当前的时间。
now = time.strftime('%H%M%S')

# 提示用户输入zip文件名中附加的注释
comment = input('输入注释 --> ')
# Check if a comment was entered
if len(comment) == 0:
    target = today + os.sep + now + '.zip'
else:
    target = today + os.sep + now + '_' + \
        comment.replace(' ', '_') + '.zip'

# 如果子目录不存在就创建它
if not os.path.exists(today):
    os.mkdir(today)
    print('成功创建子目录：', today)
```

```
# 5. 我们使用zip命令把文件压缩到一个压缩文件中
zip_command = "zip -qr {0} {1}".format(target,
                                       ' '.join(source))

# 运行备份
print("Zip命令为:")
print(zip_command)
print("运行:")
if os.system(zip_command) == 0:
    print('成功备份到', target)
else:
    print('备份失败')
```

输出：

```
$ python3 backup_ver4.py
输入注释--> added new examples
成功备份到 E:\Backup\20080702\202836_added_new_examples.zip

$ python3 backup_ver4.py
输入注释-->
成功备份到 E:\Backup\20080702\202839.zip
```

它是如何工作的：

这个程序现在工作了！让我们仔细检查第三版的实际增强，我们使用 `input` 函数获取 用户的注解，然后通过使用 `len` 函数找到输入的长度检查用户确实输入了一些东西。如果用户只是按 `enter` （回车键），没有输入任何东西(也许这只是一个常规备份或没有特殊的改变)，那么，我们按照我们之前所做的处理。

然而，如果提供了一个注释，那么，它将附加到压缩文档名字中、 `.zip` 扩展名前。请注意，我们将注释中的空格用开线正在取代空间在评论中用下划线——这是因为管理没有空格的文件名容易得多。

更细化

第四版对于大多数用户来说是一个令人满意的工作脚本，但总有改进的余地。例如，您可以为程序包括一个冗长级别，在那里你可以指定一个 `-v` 选项，从而使你的程序变得更加健谈。

另一个可能的优化处理是将允许额外的文件和目录被传递给该脚本的命令行。我们可从 `sys.argv` 列表得到这些文件名，我们可以使用 `list` 类提供的 `extend` 方法将它们添加到我们的 `source` 列表中。

最重要的改进是不使用的创建文档的 `os.system` 方式，而是使用转而使用内建的 zipfile 或 tarfile模块创建文档。他们是标准库的一部分，在你的计算机上已经为您提供使用没有外部依赖的压缩程序。

然而，在上面的例子中，纯粹是为教学的目的，我一直使用 `os.system` 的方式创建一个备份，这样的例子对每个人的理解足够简单，但不是真正足够的有效。

你能使用zipfile模块，而不是 `os.system` 调用尝试写第五版吗？

软件开发过程

我们已经经历了编写一个软件过程中的各种阶段。这些阶段可以概括如下：

1. 什么 (分析)
2. 怎样 (设计)
3. 做 (实现)
4. 测试 (测试和调试)
5. 使用 (操作和部署)
6. 维护 (优化) 编写程序的推荐方法是我们创建备份脚本的过程：做了分析和设计，开始实现用一个简单的版本，测试和调试它，来确保它能按预期工作。现在，添加你想要的任何功能，继续重复需要次数的做—试验循环。

记住：

> 软件是在成长，而不是建立。 -- Bill de hÓra

小结

我们已经看到了如何创建我们自己的Python程序/脚本和编写这种程序的不同阶段。你可能会发现创建你自己的程序就像我们在这一章做的是有用的，以便你熟悉Python以及解决问题。

接下来，我们将讨论面向对象编程。

继续阅读面向对象编程

面向对象编程

到现在为止，在我们编写的所有程序中，我们围绕着函数，也就是处理数据的语句块来设计我们的程序，这叫做面向过程的编程方式，还有一种组织你的程序的方式，是将数据和函数组合起来打包到称为对象的东西里面，这叫做面向对象编程技术。大多数情况下，你可以使用面向过程的编程方式，但当你编写大型程序或者有一些适用于这种方式更好的问题时，你可以使用面向对象的编程技术。

类和对象是面向对象编程的两个主要方面，一个类创建一个新的类型，在这里对象是类的一个实例。一个比喻，你可以有 `int` 型变量，换句话说，存储整数的变量是 `int` 类的一个实例（对象）。

> 静态语言的程序员应该注意
>
> 注意，整数甚至被看作（int类的)对象。这不像在C++和(1.5版本以前的)Java语言中整数是原始的原生数据类型
>
> 关于类的更多细节，请看help(int)。
>
> C#和Java程序员将发现这和装箱和拆封的概念相似。

对象可以使用属于对象的普通变量存储数据。属于一个对象或类的对象被称为字段。对象也可以通过使用属于类的函数有函数性。这样的函数被称为类的方法，这个术语是很重要的，因为它帮助我们区分函数和变量哪些是独立的，那些是属于一个类或对象的。总体而言，这些字段和方法可以被称为类的属性。

字段有两种类型，它们可以属于类的每个实例/对象，或属于类本身。它们被分别称为实例变量和类变量。

要创建一个类使用 `class` 的关键字，类的字段和方法在一个缩进块中列出。

`self`

类的方法与普通的函数只有一个特别的不同点--他们必须有一个额外的第一个名字、必须被添加到参数列表的开始处，但你调用该方法时，不用给此参数的值，Python将提供它。这个特别的变量指向对象本身，按照惯例，它的名字是 `self` 。

虽然,你可以给这个参数任何名字，强烈推荐你使用名称 `self` --任何其他的名字肯定是不清楚的。使用标准的名字，有许多优势--你的程序的任何读者将立即认出它，如果你使用 `self` ，甚至专门的ide(集成开发环境)也可以帮助你。

> **C++/Java/C#程序员要注意**
>
> 在Python中，`self`相当于C++中的指针`this`、Java和C#中的`this`引用。

你一定很想知道Python怎样给`self`赋值，为什么你不需要给它一个值。一个例子会使这个清楚。假设，你有一个称为`MyClass`的类和这个类的实例称为`myobject`。当你调用这个对象的方法`myobject.method(arg1, arg2)`时，Python将自动转换成`MyClass.method(myobject, arg1, arg2)`--这是关于`self`的所有特殊之处。

这也意味着,如果你有一个不带任何参数的方法，那么你还得有一个参数——`self`。

** 类

最简单的类可能是如下面的示例所示(另存为simplestclass.py).

```
class Person:
    pass # 一个空块

p = Person()
print(p)
```

输出：

```
$ python3 simplestclass.py
<__main__.Person object at 0x019F85F0>
```

它是如何工作的：

我们使用的`class`语句和类的名称创建一个新的类，接下来是形成类的主体语句的一个缩进块。在这里，我们使用pass语句表示这是一个空的块。

接下来，我们使用类名后跟一对圆括号创建这个类的一个对象/实例(在接下来的部分，我们将学习更多关于实例化的知识)。为了验证，我们通过简单地打印它确认变量的类型。它告诉我们，在`__main__`模块中有一个`Person`类的实例。

注意，你的对象存储在计算机内存的地址也被打印了。因为Python找到任何地址就存储对象，因而，在你的计算机上地址会有所不同。

对象的方法

我们已经讨论了类/对象除了有额外的self变量外，还可以有方法，就像函数。现在,我们将看到一个例子(另存为"的方法py")。

例子（保存为 oop_method.py）:

```
class Person:
    def say_hi(self):
        print('嗨，你好吗？')

p = Person()
p.say_hi()
# 上面这两行也可写成Person().say_hi()
```

输出：

```
$ python oop_method.py
嗨，你好吗？
```

它是如何工作的：

在这里我们看到 `self` 在起作用。注意， `say_hi` 方法不包含任何参数，但在函数定义中仍有 `self` 。

`__init__` 方法

在Python中有许多特别重要的方法名称，现在，我们看看 `__init__` 方法的重要性。

类的一个对象一被初始化， `__init__` 方法就运行。这个方法对你的对象做任何初始化都是有用的。

例子 (保存为 oop_init.py):

```
class Person:
    def __init__(self, name):
        self.name = name

    def say_hi(self):
        print('嗨，我的名字是', self.name)

p = Person('Swaroop')
p.say_hi()
# 以上两行也可以写成 Person('Swaroop').sayHi()
```

输出：

```
$ python class_init.py
嗨，我的名字是 Swaroop
```

它是如何工作的：

在这里，我们定义一个带参数`name`(和通常的 `self`)的`__init__`方法。在这里，我们只是创建一个新的称作`name`的字段。注意，尽管它们都叫 `name` ，但它们是两个不同的变量。因为 `self.name` 中的点符号意味着"self"对象的一部分有个叫"name" 的东西，而另一个`name`是一个局部变量,因此没有问题。因为我们明确地表明我们所指的是哪个的名字，没有混乱。

最重要的是。请注意。我们没有显式地调用 `__init__` 方法，而是当创建类的一个实例时，通过在类名称后的括号内传递参数，这是该方法的特殊意义。

现在，我们可以在我们的方法中使用`self.name`字段了，在`say_hi`方法中已经做了演示。

类和对象的变量

我们已经讨论了类与对象的部分功能(即方法)，现在让我们了解一下数据部分。数据部分，即字段，只不过是被绑定到对象和类的命名空间名字的普通变量。这意味着，这些名字只有在类和对象的环境内有效。这就是为什么他们被叫做命名空间的原因。

有两种类型的字段--类变量和对象变量，它们的分类取决于类和对象分别属于哪种变量。

类变量是共享的——他们可以被该类的所有实例访问。类变量只是一个拷贝，当任何一个对象改变一个类变量时，所有的其它实例都将改变。

对象变量是类的每个对象或实例所特有的。既然这样，每个对象都有自己的字段拷贝，也就是说，在不同的实例中，它们不共享，同名的字段没有任何联系。一个例子能使你容易理解（保存为oop_objvar.py):

```python
class Robot:
    """表示人一机器人，有一个名字。"""

    # 一个类变量，数机器人的数量
    population = 0

    def __init__(self, name):
        """初始化数据。"""
        self.name = name
        print("(初始化 {})".format(self.name))

        # 当创建一个人时，机器人人口加1
        Robot.population += 1

    def __del__(self):
        """我将要死了。"""
        print("{0} 正在被毁！".format(self.name))

        Robot.population -= 1

        if Robot.population == 0:
            print("{}是最后一个。".format(self.name))
        else:
            print("还有{:d}机器人在工作。".format(Robot.population))

    def say_hi(self):
        """机器人问候。

        是的，它们能做作那个。"""
        print("你好，我的主人叫我".format(self.name))

    @classmethod
    def how_many(cls):
        """打印当前人口。"""
        print("我们有{:d}个机器人。".format(cls.population))

droid1 = Robot('R2-D2')
droid1.say_hi()
Robot.how_many()

droid2 = Robot('C-3PO')
droid2.say_hi()
Robot.how_many()

print("\n机器人在这能做一些工作。\n")

print("机器人已经完成了它们的工作，因此，让我们销毁它们。")
droid1.die()
droid2.die()

Robot.how_many()
```

输出：

```
$ python objvar.py
(初始化 R2-D2)
你好，我的主人叫我
我们有1个机器人。
(初始化 C-3PO)
你好，我的主人叫我
我们有2个机器人。

机器人在这能做一些工作。

机器人已经完成了它们的工作，因此，让我们销毁它们。
R2-D2 正在被毁！
还有1机器人在工作。
C-3PO 正在被毁！
C-3PO是最后一个。
我们有0个机器人。
```

它是如何工作的：

这是一个很长的例子，但有助于展示类和对象变量的特性。在这里，`population` 属于 `Robot` 类，因此是一个类变量。 `name` 变量属于对象(使用 `self` 分配)，因此是一个对象变量。

因此，我们提到 `population` 类变量使用 `Robot.population` 而不是 `self.population` 。我们在那个对象的中提到对象变量 `name` 使用 `self.name` 符号。记住对象和类变量的简单区别。还请注意，一个对象变量与一个类变量名字相同时，类变量将被隐藏！

除了使用 `Robot.population` ,我们还可以使用 `self.__class__.population` 访问类变量，因为每一个对象都可以通过 `self.__class__` 属性访问他的类。

`how_many` 实际上是一个属于类而不是对象的方法，这意味着我们可以将其定义成 `classmethod` 或 `staticmethod` 中的任何一个，这取决于我们是否需要知道是哪个类。因为，我们不需要这样的信息，我们主张 `staticmethod` 。

我们使用修饰符将 `how_many` 方法标识为类方法。

我们可以把修饰符想象成为一个包装函数的快捷方式，所以使用 `@classmethod` 修饰符和下面的调用是一样的：

```
how_many = classmethod(how_many)
```

我们注意到 `__init__` 方法使用一个 `name` 变量初始化 `Robot` 实例。在这个方法中，因为还有一个机器人被添加,我们为 `population` 计数加1。还发现， `self.name` 的值是针对每一个对象的，这表明对象变量的特性。

记住,你必须只有使用 `self` 引用同一对象的变量和方法，这就是所谓的属性引用。

在这个程序中,我们也看到了类和方法的文档字符串的用法。在运行时我们可能通过使用 `Robot.__doc__` 访问类的文档字符串，使用 `Robot.say_hi.__doc__` 访问方法的为文档字符串。

在 `die` 方法中，我们简单的将 `Robot.population` 计数减1。

所有的类成员是公共的，一个例外是：如果你使用的数据成员的名字使用了 `双下划线前缀` 如 `__privatevar`, Python使用命名修饰来有效地使它成为一个私有变量。

因此,下面的惯例是，只在对象和类中使用的任何变量，首先应该以一个下划线开始，其他所有的名字都是公共的，且可以被用于其他的类/对象使用。记住，这只是一个惯例和不是被Python强制执行的(除了双下划线前缀)。

> **C++/Java/C#**程序员要注意 在Python中，所有类成员(包括数据成员)是公共有和所有的方法是虚拟。

继承

面向对象编程的一个好处是代码的重用，一种方式是通过继承机制实现，继承可以被想像为实现类之间的一种类型和子类型的关系。

假设您想编写一个大学里教师和学生记录的程序，他们有一些共同的特性，如姓名、年龄和地址。他们也有特定的特性，如老师的工资、课程和树叶和学生的学费、分数。

您可以为每个类型创建两个独立的类，并且处理它们，但要添加一个新的共同特征意味着要在这两种独立的类中都要添加，很快就会变得难以处理。

一个更好的方法是创建一个共同的类称为 `SchoolMember` ，然后从这个类继承老师类和学生类，也就是说它们成为这个类的子类，可以对这些子类添加特定的特征。

这种方式有很多优点，如果我们在 `SchoolMember` 中添加/更改任何功能，在子类中会自动反映出来。例如，您可以为学生和老师添加一个新的身份证字段，可能通过直接把它们添加到SchoolMember类中来实现。然而，子类中的变化不影响其他子类。另一个优点是，如果你引用SchoolMember类的一个老师或学生对象，在某些情况下如计算学校成员的数量时会很有用。这就是所谓的多态性，如果父类是预期的，子类在任何情况下可以被取代，即对象可以当做父类的一个实例。

还观察到，我们重用父类的代码，在不同的类中我们不需要重复，而在使用独立的类的情况下我们不得不重复。

在这种情况下， `SchoolMember` 类被称为基类或超类。 `Teacher` 和 `Student` 类被称为派生类或子类。

现在，我们将看到作为程序的这个例子(保存为 oop_subclass.py)：

```
class SchoolMember:
    '''代表任何学校成员。'''
    def __init__(self, name, age):
        self.name = name
        self.age = age
        print("(初始化学校成员： {})".format(self.name))

    def tell(self):
        '''告诉我细节。'''
        print("Name:'{}' Age:'{}'".format(self.name, self.age), end=" ")

class Teacher(SchoolMember):
    '''代表老师。'''
    def __init__(self, name, age, salary):
        SchoolMember.__init__(self, name, age)
        self.salary = salary
        print("(初始化老师： {})".format(self.name))

    def tell(self):
        SchoolMember.tell(self)
        print("Salary: '{0:d}'".format(self.salary))

class Student(SchoolMember):
    '''代表学生。'''
    def __init__(self, name, age, marks):
        SchoolMember.__init__(self, name, age)
        self.marks = marks
        print("(初始化学生： {})".format(self.name))

    def tell(self):
        SchoolMember.tell(self)
        print("Marks: '{:d}'".format(self.marks))

t = Teacher("Mrs. Shrividya", 40, 30000)
s = Student("Swaroop", 25, 75)

# 打印一个空行
print()

members = [t, s]
for member in members:
    # 为Teachers和Students工作
    member.tell()
```

输出：

```
$ python inherit.py
(初始化学校成员： Mrs. Shrividya)
(初始化老师： Mrs. Shrividya)
(初始化学校成员： Swaroop)
(初始化学生： Swaroop)

Name:"Mrs. Shrividya" Age:"40" Salary: "30000" Name:"Swaroop" Age:"25" Marks: "75"~
```

它是如何工作的：

使用继承，在类定义中，在类的名称后，我们在元组中指定基类名称，接下来，我们观察到使用 `self` 变量，显式地调用基类的 `__init__` 方法，这样我们可以初始化对象的基类部分。这是非常重要的，记住——Python不会自动调用基类的构造函数，您自己必须显式地调用它。

我们还观察到，我们可以在类名前加前缀调用基类的方法，然后和其它参数一道传递给 `self` 变量值。

注意，当我们使用 `SchoolMember` 类的 `tell` 方法时，我们可以把 `Teacher` 或 `Student` 的实例作为 `SchoolMember` 的实例。

同时，观察到子类的tell方法的调用，不是 `SchoolMember` 类的 `tell` 方法。要理解这一点的一种方法是，Python 总是在实际的类型中开始寻找方法，如本例。如果它不能找到方法，它开始按在类定义中元组中指定的顺序一个接一个地查找属于它的基类的方法。

术语提示--如果在继承元组中不止列出一个类，那么它被称为多重继承。

在 `tell()` 方法中， `end` 参数是用于来将换行变为在 `print()` 调用结束后以空格开始。

小结

我们已经探讨了类和对象的各个方面以及与之关联的各种术语。我们也看到了面向对象编程的好处和缺陷。Python是高度面向对象，从长远看仔细理解这些概念仔细将对你很有帮助。

接下，我们将学习如何处理输入/输出和如何在Python中访问文件。

继续阅读输入/输出

输入 输出

会有这种情况，你的程序必须与用户进行交互。例如，你想获取来自用户的输入，然后打印一些返回的结果。我们可以分别使用input()和print()函数来实现。

对于输出，我们还可以使用str(字符串)类的各种方法。例如，您可以使用rjust方法来获取一个指定宽度的字符串。更多细节，见 help(str)。

另一个常见的输入/输出类型是处理文件。创建、读和写文件是许多程序至关重要的，我们将在本章探讨这方面。

用户输入

将这个程序保存为 user_input.py:

```
def reverse(text):
    return text[::-1]

def is_palindrome(text):
    return text == reverse(text)

something = input('输入文本： ')
if (is_palindrome(something)):
    print("是的，这是回文")
else:
    print("不，这不是回文")
```

输出：

```
输入文本： 蜜蜂
不，这不是回文
输入文本： 人上人
是的，这是回文
```

它是如何工作的：

我们使用切片特性来颠倒文本。我们已经看到使用seq[a:b]代码获取从a到b来自序列的切片。我们还可以提供一个第三个确定步长的参数，切片默认的步长是 1，它返回一个连续文本的一部分。给一个负的步长，即 -1 ，将以反向返回文本。

input()函数将一个字符串作为参数，并显示给用户。然后等待用户输入和按回车键。一旦用户输入和按下回车键，input()函数将返回用户输入的文本。

我们获取文本并颠倒它。如果原始文本和颠倒的文本是相等的，那么那个文本是一个回文。

家庭作业

> 检查一个文本是否是一个回文应该忽略标点符号、空格和案例。例如，"Rise to vote, sir." 也是一个回文，但我们当前的程序并没有说它是。你能改善上述程序来识别这个回文吗?

下面的提示(不要读)

> 使用一个元组(从这里(http://grammar.ccc.commnet.edu/grammar/marks/marks.htm)你可以找到所有标点符号的一个列表)来保存所有的禁止字符，然后使用会员测试，以确定是否应该删除一个字符，即forbidden = ('!', '?', '.', ...)。

文件

为了读写，你可以通过创建一个file类的对象，分别使用read、readline或 write方法来，打开和使用文件。能够读取或写入文件取决于文件打开时指定的模式。最后，当你完成对文件的操作时，你要调用close方法告诉Python，文件我们使用完了。

例子 (保存为 using_file.py):

```
poem = '''\
当工作完成时
编程是有趣的
如果想让你的工作有趣
    使用Python！
'''

f = open('poem.txt', 'w') # 为'写w'打开文件
f.write(poem) # 文本写入文件
f.close() # 关闭文件

f = open('poem.txt') # 如果不指定打开模式，默认为'读'
while True:
    line = f.readline()
    if len(line) == 0: # 0长度表示文件结尾
        break
    print(line, end='')
f.close() # 关闭文件
```

输出：

```
D:> python using_file.py
当工作完成时
编程是有趣的
如果想让你的工作有趣
    使用Python！
```

它是如何工作的：

首先，通过内置的函数open，指定文件名和我们要打开的模式，打开一个文件。模式可以是读模式('r'), 写模式('w')或追加模式('a')。我们也可以指定是否以文本格式('t') 或二进制格式('b')读,写或追加。实际上有更多可用的模式，help(open) 会给你更多的细节。默认情况下，open()认为是一个以读方式打开的文本格式的文件。

在我们的例子中，我们首先以写文本格式打开文件，使用文件对象的write方法写文件，然后,我们最后 close(关闭)文件。

接下来，为再次阅读，我们打开同一个文件。我们不需要指定一个模式,因为 '读文本文件' 是默认的模式。我们使用readline方法在一个循环中每次读文件的一行。该方法返回一个完整的行，包括换行符结束时的行。当返回一个空字符串时，这意味着我们已经到达文件的末尾，我们'打破'循环。

在默认情况下，print()函数在屏幕上自动换行打印文本。我们是通过指定end=''禁止产生新行，因为从文件读取的行在结尾已经包含一个换行符。然后，我们最终close文件。

现在，检查poem.txt的内容，确认程序确实写入和从那个文件读取。

拾取

Python提供了一个标准的模块称为pickle，使用它你可以在一个文件中存储任何的Python对象，然后把它弄回来后，这就是所谓的持续的存储对象。

例子 (保存为 pickling.py):

```
import pickle

# 我们将要存储对象的文件名
shoplistfile = 'shoplist.data'
# 购物清单
shoplist = ['苹果', '芒果', '胡萝卜']

# 定到文件
f = open(shoplistfile, 'wb')
pickle.dump(shoplist, f) # 把对象倒入一个文件
f.close()

del shoplist # 释放shoplist变量

# 从仓库读回
f = open(shoplistfile, 'rb')
storedlist = pickle.load(f) # 从文件载入对象
print(storedlist)
```

输出：

```
D:> python pickling.py
['苹果', '芒果', '胡萝卜']
```

它是如何工作的：

要在文件中存储一个对象，我们首先必须以'w'rite写'b'inary 二进制格式的方式open打开文件，然后调用pickle模块的dump函数，这个过程叫拾取。

接下来，我们使用pickle模块的load函数取回对象，这个过程叫做拆开。

小结

我们已经讨论了各种类型的输入/输出，文件处理和使用pickle模块。

接下来，我们将探讨索异常的概念。

继续阅读异常

异常

当你的程序处于异常的状态的时候，会抛出异常。例如当你想要读取一个并不存在的文件的时候，或者当你要删除一个正在运行的程序的时候。这些情况通过异常来处理。

类似的，如果你的程序有一些无效的语句，Python也会抛出错误提示告诉你这里有一些错误。

错误

我们来看一下一个简单的 `print` 函数。如果我们把 `print` 写成了 `Print` 会怎样？注意大小写的错误。这是Python会抛出一个语法错误。

```
>>> Print("Hello World")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'Print' is not defined
>>> print("Hello World")
Hello World
```

我们注意到抛出了一个 `NameError` 的错误，以及这个错误发生的位置。这就是当错误发生的时候错误处理程序所做的事情。

异常

我们尝试从控制台读取用户输入的信息，然后按下 `[ctrl-d]` 看看会发生什么。

```
>>> s = input('请输入 --> ')
Enter something --> Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
EOFError
```

Python抛出了一个名为 `EOFError` 的错误信息，他的是*end of file*的缩写(由 `ctrl-d` 触发)，这是我们的程序刚开始的时候没有预料到的。

异常处理

我们可以用 `try..except` 语句处理异常。我们将正常执行的语句放在try语句块中，然后将错误处理程序放到except语句块中。

例如 (保存为 `exceptions_handle.py`):

```
try:
    text = input('请输入 --> ')
except EOFError:
    print('为什么你按下了EOF？')
except KeyboardInterrupt:
    print('你取消了操作')
else:
    print('你输入了 {}'.format(text))
```

输出为:

```
# 按下 ctrl + d
$ python exceptions_handle.py
请输入 --> 为什么你按下了EOF？

# Press ctrl + c
$ python exceptions_handle.py
请输入 --> ^C你取消了操作

$ python exceptions_handle.py
请输入 --> No exceptions
你输入了 No exceptions
```

它是如何工作的：

我们将所有的可能会抛出异常/错误的语句写在 `try` 块中，然后将对应的处理程序写在 `except` 块中。每个 `except` 语句可以处理一个特定的异常/错误，或者是一个异常/错误的列表（用括号表示）。如果没有指明异常/错误的名字，那么他会处理所有的错误/异常。

注意，每一个 `try` 语句至少应该有一个与之匹配的 `except` 语句，否则try语句就没有意义了。

如果你的程序发生了异常/错误，但是没有被处理，那么Python语言就会启动默认的异常处理程序，它会中止程序的运行，打印出错误的信息，这些内容我们已经看到了。

你也可以给你的 `try..except` 写上一个 `else` 语句块，当没有任何异常发生的时候就会执行 `else` 语句的内容。

在下面的例子中，我们将会学习如何获得异常对象，以便于我们能够得到关于异常的更多的信息。

抛出异常

你可以使用 `raise` 语句抛出一个异常，在语句中你需要提供异常/错误的名称以及抛出的异常对象。

你抛出的异常/错误必须是一个从 `Exception` 派生的类。

例如：(保存为 `exceptions_raise.py`)：

```
class ShortInputException(Exception):
    '''用户自定义的异常类。'''
    def __init__(self, length, atleast):
        Exception.__init__(self)
        self.length = length
        self.atleast = atleast

try:
    text = input('请输入 --> ')
    if len(text) < 3:
        raise ShortInputException(len(text), 3)
    # Other work can continue as usual here
except EOFError:
    print('Why did you do an EOF on me?')
except ShortInputException as ex:
    print(('ShortInputException: The input was ' +
           '{0} long, expected at least {1}')
          .format(ex.length, ex.atleast))
else:
    print('No exception was raised.')
```

输出为：

```
$ python exceptions_raise.py
请输入 --> a
ShortInputException: The input was 1 long, expected at least 3

$ python exceptions_raise.py
请输入 --> abc
No exception was raised.
```

它是如何工作的：

在这里我们创建了我们自己的异常类。新的异常类为 `ShortInputException` 。他有两个字段： `length` 表示输入内容的长度， `atleast` 表示程序期望的最小长度。

在 `except` 语句中，我们制定由 `as` 变量保存弹出的异常/错误的对象。这很类似函数参数在函数调用中的作用。在这个特殊的 `except` 语句中，我们使用异常对象的 `length` 和 `atleast` 字段构造了一个异常提示信息，让用户了解为什么会抛出这个异常。

Try ... Finally

设想一下你的程序需要读取一个文件，你怎样保证无论是否有异常抛出，文件对象都被正确的关闭呢？我们可以使用 `finally` 语句块做到这一点。

例如：（保存为 `exceptions_finally.py` ）

```
import sys
import time

f = None
try:
    f = open("poem.txt")
    # Our usual file-reading idiom
    while True:
        line = f.readline()
        if len(line) == 0:
            break
        print(line, end='')
        sys.stdout.flush()
        print("Press ctrl+c now")
        # To make sure it runs for a while
        time.sleep(2)
except IOError:
    print("Could not find file poem.txt")
except KeyboardInterrupt:
    print("!! You cancelled the reading from the file.")
finally:
    if f:
        f.close()
    print("(Cleaning up: Closed the file)")
```

输出为：

```
$ python exceptions_finally.py
Programming is fun
Press ctrl+c now
^C!! You cancelled the reading from the file.
(Cleaning up: Closed the file)
```

它是如何工作的：

我们读取文件的内容，只是每读一行就让系统休息2秒，我们使用 `time.sleep` 函数让程序运行慢一点（正常情况下Python程序运行的飞快）。当程序还在运行的时候，按下 `ctrl + c` 键中止程序的运行。

我们注意到当程序退出的时候抛出了 `KeyboardInterrupt` 异常。然而，在程序退出之前，执行了finally语句块，并且文件对象被正确的关闭了。

注意，我们在 `print` 函数后面调用 `sys.stdout.flush()` 函数，这样可以及时输出结果。

with语句

在 `try` 语句块中获取资源，然后再 `finally` 语句块中释放资源是一个非常常用的程序段，我们可以使用 `with` 简化一下程序的书写。

例如：（保存为 `exceptions_using_with.py` ）

```
with open("poem.txt") as f:
    for line in f:
        print(line, end='')
```

它是如何工作的：

这段程序的输出应该和之前的例子是一模一样的。唯一的区别在与我们在 `with` 语句中使用 `open` 函数打开文件，这样的话系统会自动关闭这个文件。

实际的处理过程是这样的， `with` 语句会获取 `open` 函数返回的对象，我们假定这个对象名称是"thefile"。

它总是会在进入 `with` 语句块之前调用 `thefile.__enter__` 函数，并且总是会在语句块的最后调用 `thefile.__exit__` 函数。

这样的话我们之前在 `finally` 语句块中写的程序就会自动的在 `__exit__` 方法中被执行，这种方式可以防止我们频繁使用 `try..finally` 语句。

关于这个主题更多的讨论已经超出了本书的范畴，请参考PEP 343。

总结

本章我们讨论了 `try..except` 和 `try..finally` 语句，我们还自己定义了一个我们自己的异常类型，并且在程序中将其抛出。

下一步，我们将会浏览一下Python标准库。

继续阅读标准库

标准库

Python标准库包括大量有用的模块，安装完Python之后就随之安装了。Python标准库能够帮助你快速解决很多问题，如果你非常熟悉这些库可以做什么。因此熟练掌握Python标准库非常重要。

我们会快速浏览一下最常用的标准库模块，如果你想查看Python标准库的完整文档，请访问标准库参考，这份文档随着Python安装包也安装在你的电脑上。

让我们开始浏览一些有用的库模块。

> 注意：如果你发现本章讨论的问题太过深奥，你可以略过本章。然而，我强烈建议你在熟练掌握Python编程技能之后再返回头过来看看本章的内容。

`sys` 模块

`sys` 模块包括一些与操作系统相关的功能。我们已经知道了 `sys.argv` 可以列出命令行的参数列表。

假定我们想要检查我们使用的Python版本，我们可以使用 `sys` 模块完成这个工作。

```
>>> import sys
>>> sys.version_info
sys.version_info(major=3, minor=5, micro=1, releaselevel='final', serial=0)
>>> sys.version_info.major == 3
True
```

它是如何工作的

`sys` 模块包含了一个名为 `version_info` 的元组，保存着Python软件的版本信息，其中第一项为主版本号， 我们可以通过读取这些内容来使用它。

日志模块

如果你想要输出调试信息，或者保存其他一些重要信息，以便于你可以随时调取用来检查你的程序是否按照你的想法在运行。怎样做才能保存这些信息呢？可以使用 `logging` 模块。

保存为 `stdlib_logging.py` ：

```
import os
import platform
import logging

if platform.platform().startswith('Windows'):
    logging_file = os.path.join(os.getenv('HOMEDRIVE'),
                                os.getenv('HOMEPATH'),
                                'test.log')
else:
    logging_file = os.path.join(os.getenv('HOME'),
                                'test.log')

print("Logging to", logging_file)

logging.basicConfig(
    level=logging.DEBUG,
    format='%(asctime)s : %(levelname)s : %(message)s',
    filename=logging_file,
    filemode='w',
)

logging.debug("Start of the program")
logging.info("Doing something")
logging.warning("Dying now")
```

输出为：

```
$ python stdlib_logging.py
Logging to /Users/swa/test.log

$ cat /Users/swa/test.log
2014-03-29 09:27:36,660 : DEBUG : Start of the program
2014-03-29 09:27:36,660 : INFO : Doing something
2014-03-29 09:27:36,660 : WARNING : Dying now
```

如果你没有 `cat` 命令，你可以使用一个Visual Studio Code打开 `test.log` 。

它是如何工作的

在这个程序中我们使用了Python标准库的三个模块： `os` 模块用于和操作系统进行交互， `platform` 模块用于获取运行平台（如操作系统）的信息， `logging` 模块用于记录日志。

首先，我们调用 `platform.platform()` 检查程序运行的操作系统（更详细的信息请参考 `import platform; help(platform)` ）。如果是Windows操作系统，我们通过指定主盘、主目录和文件名保存信息。我们把这三个信息拼接起来，就得到了日志文件的完整路径。针对其他的操作系统，我们只需要知道用户的主目录以及文件名就可以得到日志文件的完整路径。

我们使用 `os.path.join()` 函数连接三部分作为一个字符串，保存日志文件的完整路径。之所以使用函数做字符串连接而不是采用字符串 `+` 操作，是因为我们要确保这个操作可以在所有的操作系统环境下运行。

我们配置了 `logging` 模块，把所有的信息用特定的格式写到我们指定的文件中。

最后，我们可以指定输出的信息的属性，比如调试、提示、警告或者致命错误。程序运行起来之后，我们可以通过查看这个文件了解我们的程序的运行情况，即使我们的程序没有向用户输出信息。

Python Module of the Week

还有很多其他有用的模块，比如调试, 处理命令行参数, 正则表达式等等，已经超出了本书的范围。

想要继续学习Python标准库，推荐大家阅读Doug Hellmann写的非常棒的Python Module of the Week，也可以在Amazon购买纸质的书籍，当然你也可以直接阅读Python官方文档。

总结

我们浏览了Python标准库的一些模块，强烈推荐大家快速浏览一下Python标准库官方文档，以便知道有哪些模块时可以拿来就用的。

接下来，我们探讨一下Python语言其他方面的事情，这样可以让我们这本教程更加完整。

继续阅读更多

更多

迄今为止我们已经学习了Python中的大多数常用知识。本章中我们会接触到更多的知识，使得我们更全面的掌握Python。

传递元组

你是否希望过从函数返回两个不同的值？做到这点使用元组即可。

```
>>> def get_error_details():
...     return (2, 'details')
...
>>> errnum, errstr = get_error_details()
>>> errnum
2
>>> errstr
'details'
```

注意，我们使用 `a, b = <some expression>` 这个表达式把元组的两个字段分别赋给两个变量。

这也意味着在Python中最快速的交换两个变量的值得方法是：

```
>>> a = 5; b = 8
>>> a, b
(5, 8)
>>> a, b = b, a
>>> a, b
(8, 5)
```

特殊方法

有一些诸如**intit**和**del**的方法在类中拥有特殊的含义。

特殊方法用于模拟某些内建类型的行为。例如，你希望为你的类使用 `x[key]` 索引操作(就像在列表和元组中那样)，那么你仅仅需要实现 `__getitem__` 方法就可以了。顺便提一句，Python正是这样实现 `list` 类的！

一些有用的特殊方法列在下表中。如果你想了解所有的特殊方法，详见帮助文档.

- `__init__(self, ...)`

    - 在对象第一次被创建后，返回之前调用。
- `__del__(self)`
    - 在对象被销毁前调用（我们无法预期这个函数什么时候被调用，因此尽量避免使用它）。
- `__str__(self)`
    - 在使用 `print` 函数或 `str()` 时调用。
- `__lt__(self, other)`
    - 在使用小于运算符时(<)调用。类似的其它运算符也存在对象的特殊方法(+, >等)。
- `__getitem__(self, key)`
    - 当使用 `x[key]` 索引操作时调用。
- `__len__(self)`
    - 当使用内建 `len()` 函数时调用，一般用于序列的对象。

单语句块

我们已经看到每个语句块都根据它的缩进级别将彼此区分开。不过有一个例外，如果某语句块只包含单条语句，你可以把它放到同一行，例如条件语句或循环语句。下面的例子清楚的说明了这点：

```
>>> flag = True
>>> if flag: print('Yes')
...
Yes
```

注意上面的单条语句被放置到同一行而没有作为单独的块。虽然你利用这点可以让程序变的更短，但我强烈建议你避免使用这个快捷方式(除了错误检测)，主要原因是使用适当的缩进可以更方便的添加额外的语句。

Lambda表达式

`lambda` 语句用于在运行时创建新的函数对象。通常情况下 `lambda` 语句带有一个参数，后面跟着一个简单的表达式作为函数体，把参数代入函数得到的返回值就是新的函数的返回值。

例如 (保存为 `more_lambda.py`):

```
points = [{'x': 2, 'y': 3},
          {'x': 4, 'y': 1}]
points.sort(key=lambda i: i['y'])
print(points)
```

输出:

```
$ python more_lambda.py
[{'y': 1, 'x': 4}, {'y': 3, 'x': 2}]
```

它是如何工作的：

注意，`list` 对象的 `sort` 函数有一个名为 `key` 的参数决定了这个列表是怎样被排序的（通常情况下为升序或者降序）。在我们的例子中，我们想要有一个自己的排序规则，我们需要写一个比较函数，而不是使用 `def` 定义一个只在这里使用一次的函数，因此我们使用lambda表达式创建一个新的函数。

列表解析

列表解析用于从一个现有的列表派生出一个新的列表。 假设你有一个数字列表，你想让其中所有大于2的元素乘以2并组成一个新的列表。类似问题正是使用列表解析的理想场合。

例子 (保存为 `more_list_comprehension.py`):

```
listone = [2, 3, 4]
listtwo = [2*i for i in listone if i > 2]
print(listtwo)
```

输出:

```
$ python more_list_comprehension.py
[6, 8]
```

它是如何工作的：

当某些条件满足时(`if i > 2`)我们执行某些操作(`2 * i`)，由此产生一个新列表。注意原始列表并不会被改变。

使用列表解析的好处在于，当我们使用循环遍历元素并将其存储到新列表时可以减少样板代码量。

函数接收元组和列表

这里有一种特殊的方法可以将函数的形参当做元组或字典，那就是分别使用 `*` 和 `**` 前缀。当需要在函数内得到可变数量的实参时这个方法很有用。

```
>>> def powersum(power, *args):
...     '''Return the sum of each argument raised to the specified power.'''
...     total = 0
...     for i in args:
...         total += pow(i, power)
...     return total
...
>>> powersum(2, 3, 4)
25
>>> powersum(2, 10)
100
```

因为 `args` 变量带有 `*` 前缀，因此所有额外的实参都会被当做一个元组存入 `args` 中并传给函数。如果把这里的 `*` 换成 `**` ，则所有额外的形参都会被当做一个字典的键/值对。

assert语句

`assert` 用于断言一个表达式为真。例如，你需要确保正在使用的列表至少有一个元素，否则引发一个错误，这种情况很适合使用 `assert` 语句。当assert语句断言失败，则引发一个 `AssertError` 。

```
>>> mylist = ['item']
>>> assert len(mylist) >= 1
>>> mylist.pop()
'item'
>>> assert len(mylist) >= 1
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AssertionError
```

`assert` 应当慎重使用。多数时候用于捕获异常，处理问题或是向用户显示错误后随即终止程序。

装饰器(decorator)

装饰器是包装函数的一种快捷方式。如果你有很多函数使用了同样的一段代码，使用装饰器对函数进行包装会非常方便。例如，我创建了一个 `retry` 装饰器，可以用在任何函数中，当触发任何异常的时候，他会让这个函数不断重复执行，每次执行之间有一个固定的间隔，最多执行5次。这种设计在需要通过网络进行远程调用的时候非常有用：

```
from time import sleep
from functools import wraps
import logging
logging.basicConfig()
log = logging.getLogger("retry")


def retry(f):
    @wraps(f)
    def wrapped_f(*args, **kwargs):
        MAX_ATTEMPTS = 5
        for attempt in range(1, MAX_ATTEMPTS + 1):
            try:
                return f(*args, **kwargs)
            except:
                log.exception("Attempt %s/%s failed : %s",
                              attempt,
                              MAX_ATTEMPTS,
                              (args, kwargs))
                sleep(10 * attempt)
        log.critical("All %s attempts failed : %s",
                     MAX_ATTEMPTS,
                     (args, kwargs))
    return wrapped_f


counter = 0


@retry
def save_to_database(arg):
    print("Write to a database or make a network call or etc.")
    print("This will be automatically retried if exception is thrown.")
    global counter
    counter += 1
    # This will throw an exception in the first call
    # And will work fine in the second call (i.e. a retry)
    if counter < 2:
        raise ValueError(arg)


if __name__ == '__main__':
    save_to_database("Some bad value")
```

输出:

```
$ python more_decorator.py
Write to a database or make a network call or etc.
This will be automatically retried if exception is thrown.
ERROR:retry:Attempt 1/5 failed : (('Some bad value',), {})
Traceback (most recent call last):
  File "more_decorator.py", line 14, in wrapped_f
    return f(*args, **kwargs)
  File "more_decorator.py", line 39, in save_to_database
    raise ValueError(arg)
ValueError: Some bad value
Write to a database or make a network call or etc.
This will be automatically retried if exception is thrown.
```

它是如何工作的：

请参考:

- http://www.ibm.com/developerworks/linux/library/l-cpdecor.html
- http://toumorokoshi.github.io/dry-principles-through-python-decorators.html

Python 2 与Python 3的区别

请参考:

- "Six" library
- Porting to Python 3 Redux by Armin
- Python 3 experience by PyDanny
- Official Django Guide to Porting to Python 3
- Discussion on What are the advantages to python 3.x?

Summary

本章我们探讨了Python语言更多的特性。虽然我们仍然没有覆盖到Python语言的全部特性，但基本上已经可以应付在实践中的绝大多数情况。对于你即将创建的任何应用程序来说这就已经足够了。

接下来，我们来看看阅读完本书之后怎样继续学习Python。

继续阅读继续学习

继续学习

如果你有认真通读本书之前的内容并且实践其中包含的大量例程，那么你现在一定可以熟练使用python了。 同时你可能也编写了一些程序用于验证python特性并提高你的python技能。如果还没有这样做的话，你应该去试试。 现在的问题是接下来应该做什么？

我建议你先解决下面的问题：

> 创建你自己的命令行版本的通讯录程序，利用它你可以浏览修改删除或搜索诸如朋友，家人，同事等联系人和他们的email地址/或电话号码等信息。这些信息必须存起来以便需要时提取。

思考下我们已经学到的各种知识，这个问题其实相当简单。 如果你感觉还是不好下手的话，这有一些提示。

创建一个表示联系人(persion)信息的类。使用字典存储联系人对象并以人物的名字作为字典键。 然后利用pickle模块把这些对象永久存储到你的硬盘中。最后通过字典的内建方法add, delete和modify分别增加删除修改联系人。

只要你有能力完成这个程序，你就可以自信的说你是一个python程序员了。那么现在马上给我发送e-mail好感谢我编写了如此强大的教程吧;-)当然这步是可选的但我还是希望你发过来。同时，也请考虑下购买纸质书籍以支持本书的持续发展。

如果你觉得上面的程序太简单，这还有另一个：

> 实现replace命令。此命令用于在给定的文件列表中的所有文件中替换指定的字符串。

replace命令可以简单的执行字符串替换也可以复杂的进行模式查找(正则表达式)，这取决于你的意愿。

继续完成新的项目

如果你发现上述程序依然很简单，那么你可以看一下这个更加复杂的项目清单，用你自己的方法完成这些程序： https://github.com/thekarangoel/Projects#numbers (同样的清单还可以参考 Martyr2's Mega Project List).

更多的项目：

- Exercises for Programmers: 57 Challenges to Develop Your Coding Skills
- Intermediate Python Projects.

实例代码

学习程序设计最好的办法就是编写阅读大量代码：

- Python Cookbook 对于某些种类的问题Python Cookbook提供了许多解决问题的珍贵技巧和诀窍。此网是每个python用户都必读的。
- Python Module of the Week 是另外一个必读的网站，主要是Python标准库的指南。

忠告

- The Hitchhiker's Guide to Python!
- The Elements of Python Style
- Python Big Picture
- "Writing Idiomatic Python" ebook (paid)

视频

- Full Stack Web Development with Flask
- PyVideo

问题与解答

- Official Python Dos and Don'ts
- Official Python FAQ
- Norvig's list of Infrequently Asked Questions
- Python Interview Q & A
- StackOverflow questions tagged with python

教程

- Hidden features of Python
- What's the one code snippet/python trick/etc did you wish you knew when you learned python?
- Awaretek's comprehensive list of Python tutorials

讨论组

如果你被某个问题难住了，也不知道找谁求助，那么python-tutor list是个提问的好地方。

提问之前需要先做做功课，你应该自己先尝试解决问题，然后智慧的提问

新闻

如果你想了解python的最新动态，请关注Official Python Planet。

安装库

Python包索引Python Package Index拥有数量巨大的开源库，你可以在自己的程序中使用它们。

安装和使用这些库，你可以使用pip工具。

创建一个网站

你可以使用Django创建你自己的网站，本文后面会教你一步一步学习使用Django Web框架)。（我自己的喜好，原文写的是Flask框架，但是这个框架动不动就好几个月都不更新，我想还是选择一个由团队持续维护的框架比较靠谱）

编写GUI应用程序

如果你想使用Python创建自己的GUI应用程序。那么可以使用已绑定到Python上的GUI(图形用户界面)库。 绑定允许你在自己的程序中使用这些库，而库本身是用C/C++或其它语言编写的。

使用Python你可以选择很多种GUI库：

- Kivy
  - http://kivy.org
- PyGTK
  - GTK+工具包的python绑定。它是GNOME的基础。GTK+含有很多奇怪的用法，不过一旦熟悉它你就能够快速创建GUI应用了。其中Glade图形界面设计器是必不可少的。GTK+的文档仍然完善中。GTK+在linux上工作的很好，但其windows实现仍未完成。 另外使用GTK+你既可以创建开源也可以创建私有软件。 入门可以阅读PyGTK教程.
- PyQt

   - 这是绑定到Python的Qt工具包，它是创建KDE的基石。 Qt非常易用，功能又很强大，尤其是仰仗于它的Qt Designer与出色的Qt文档。 如果你在创建开源软件(GPL'ed)则PyQt是免费的, 相反创建商业的私有软件的用户就要掏银子买它了。 从Qt4.5开始你同样可以用它创建非GPL软件。 作为入门可以阅读PyQt5从入门到精通。
- wxPython
   - 这是绑定到python的wxWidgets工具包。 wxPython有一定的学习曲线。但是具有很强的可移植性，可以运行在linux，windows，Mac甚至是嵌入式平台之上。wxPython拥有很多可用的IDE，其中包括GUI设计器和诸如SPE(Stani的python编辑器)(http://spe.pycs.net)和wxGlade(http://wxglade.sourceforge.net/)的开发工具。 使用wxPython你既可以创建开源软件也可以创建私有软件。入门可以阅读wxPython教程(http://zetcode.com/wxpython/)

GUI小结

更多的选择参见GuiProgramming wiki page at the official python website。

很不幸，python没有一个标准GUI工具。我建议根据你的情况选择上面的工具。考虑的第一歌因素是你是否愿意付费使用GUI工具。第二你是否希望程序只运行在windows或mac或linux还是希望都能运行。第三对于linux平台，你是一个KDE还是一个GNOME用户呢。

【译者】：对于国内用户来说，首选PyQt5平台，首先Qt工具是这几个工具中使用最广泛的，有一些知名的公司如LG、松下、ABB等都采用Qt作为GUI开发平台；其次，PyQt是这几个工具中升级更新最频繁的，选用一个长期不更新的软件包会各种奇葩的坑等着你去经历；最后，Qt和PyQt的中文的开发文档以及社区也是最完整的。还有，GPL版权（此处省去23412字）。

更详细广泛的分析，见Python Papers 第26页 卷3问题1(http://archive.pythonpapers.org/ThePythonPapersVolume3Issue1.pdf)

各种python实现

一个程序设计语言通常包含两部分 – 语言和软件。语言指出如何编写程序。而软件用来运行我们的程序。

我们一直在用CPython运行我们的程序，之所以称为CPython是因为它是用C语言实现的并且为标准Python解释器。

另外还有其它的软件也可以运行python程序：

- Jython

  - 一个运行在Java平台的Python实现。这意味着你可以在Python语言内部使用Java库和类，反之亦然。
- IronPython
  - 一个运行在.NET平台的Python实现。即你可以在Python语言内部使用.NET库和类，反之亦然。
- PyPy
  - 一个用python写的python实现！这是一个研究项目，用于使之可以快而容易的改进解释器，因为解释器本身就是用动态语言编写的。（而不是类似上面的C, java或C#等静态语言）

除此之外还有CLPython(http://common-lisp.net/project/clpython/)一个Common Lisp编写的python实现。Brython是一个运行在JavaScript解释器之上的IronPython的接口，这可能意味着你可以使用python(替代JavaScript)编写web浏览器程序("Ajax")。

以上的每个实现都有自己的擅长领域。

函数式编程 (为更高级别用户)

当你要开始完成较大规模的应用程序的时候，你一定要学习一下函数式编程范式，这是与我们前面所学的面向对象编程相对应的另外一种编程范式：

- Functional Programming Howto by A.M. Kuchling
- Functional programming chapter in 'Dive Into Python' book
- Functional Programming with Python presentation
- Funcy library
- PyToolz library

Summary

现在我们已经来到本书的结尾了。不过据说，结束意味着另一个开始！你现在是一个满腔热情的Python程序员，很可能摩拳擦掌准备利用Python解决大量问题。现在你可以让计算机自动完成许多以前无法想象的事情或是编写游戏或是更多更多。既然如此！那就行动起来大干一场吧！

继续阅读附录：免费/自由和开放源码软件

附录：免费/自由和开放源码软件

> 注意：本章节是在2003年完成的，所以有些内容看起来可能有些奇怪 :-)

"Free/Libre and Open Source Software", in short, FLOSS is based on the concept of a community, which itself is based on the concept of sharing, and particularly the sharing of knowledge. FLOSS are free for usage, modification and redistribution.

If you have already read this book, then you are already familiar with FLOSS since you have been using *Python* all along and Python is an open source software!

Here are some examples of FLOSS to give an idea of the kind of things that community sharing and building can create:

Linux: This is a FLOSS OS kernel used in the GNU/Linux operating system. Linux, the kernel, was started by Linus Torvalds as a student. Android is based on Linux. Any website you use these days will mostly be running on Linux.

Ubuntu: This is a community-driven distribution, sponsored by Canonical and it is the most popular GNU/Linux distribution today. It allows you to install a plethora of FLOSS available and all this in an easy-to-use and easy-to-install manner. Best of all, you can just reboot your computer and run GNU/Linux off the CD! This allows you to completely try out the new OS before installing it on your computer. However, Ubuntu is not entirely free software; it contains proprietary drivers, firmware, and applications.

LibreOffice: This is an excellent community-driven and developed office suite with a writer, presentation, spreadsheet and drawing components among other things. It can even open and edit MS Word and MS PowerPoint files with ease. It runs on almost all platforms and is entirely free, libre and open source software.

Mozilla Firefox: This is *the* best web browser. It is blazingly fast and has gained critical acclaim for its sensible and impressive features. The extensions concept allows any kind of plugins to be used.

Mono: This is an open source implementation of the Microsoft .NET platform. It allows .NET applications to be created and run on GNU/Linux, Windows, FreeBSD, Mac OS and many other platforms as well.

Apache web server: This is the popular open source web server. In fact, it is *the* most popular web server on the planet! It runs nearly more than half of the websites out there. Yes, that's right - Apache handles more websites than all the competition (including Microsoft IIS) combined.

VLC Player: This is a video player that can play anything from DivX to MP3 to Ogg to VCDs and DVDs to ... who says open source ain't fun? ;-)

This list is just intended to give you a brief idea - there are many more excellent FLOSS out there, such as the Perl language, PHP language, Drupal content management system for websites, PostgreSQL database server, TORCS racing game, KDevelop IDE, Xine - the movie player, VIM editor, Quanta+ editor, Banshee audio player, GIMP image editing program, ... This list could go on forever.

To get the latest buzz in the FLOSS world, check out the following websites:

- OMG! Ubuntu!
- Web Upd8
- DistroWatch
- Planet Debian

Visit the following websites for more information on FLOSS:

- GitHub Explore
- Code Triage
- SourceForge
- FreshMeat

So, go ahead and explore the vast, free and open world of FLOSS!

继续阅读附录: 版本历史

附录：关于

Almost all of the software that I have used in the creation of this book are FLOSS.

Birth of the Book

In the first draft of this book, I had used Red Hat 9.0 Linux as the foundation of my setup and in the sixth draft, I used Fedora Core 3 Linux as the basis of my setup.

Initially, I was using KWord to write the book (as explained in the history lesson).

Teenage Years

Later, I switched to DocBook XML using Kate but I found it too tedious. So, I switched to OpenOffice which was just excellent with the level of control it provided for formatting as well as the PDF generation, but it produced very sloppy HTML from the document.

Finally, I discovered XEmacs and I rewrote the book from scratch in DocBook XML (again) after I decided that this format was the long term solution.

In the sixth draft, I decided to use Quanta+ to do all the editing. The standard XSL stylesheets that came with Fedora Core 3 Linux were being used. However, I had written a CSS document to give color and style to the HTML pages. I had also written a crude lexical analyzer, in Python of course, which automatically provides syntax highlighting to all the program listings.

For the seventh draft, I was using MediaWiki as the basis of my setup. I used to edit everything online and the readers can directly read/edit/discuss within the wiki website, but I ended up spending more time fighting spam than writing.

For the eight draft, I used Vim, Pandoc, and Mac OS X.

For the ninth draft, I switched to AsciiDoc format and used Emacs 24.3, tomorrow theme, Fira Mono font and adoc-mode to write.

Now

2016: I got tired of several minor rendering issues in AsciiDoctor, like the `++` in `C/C++` would disappear and it was hard to keep track of escaping such minor things. Plus, I had become reluctant to edit the text because of the complex Asciidoc format.

For the tenth draft, I switched to writing in Markdown + GitBook format, using the Spacemacs editor.

About the Author

See

继续阅读附录: 关于

附录：版本历史

我为我编写的“Diamond”软件编写简化安装过程的安装程序时，我第一次开始使用Python。我不得不在Python还是Perl上绑定Qt库进行选择。我在网上做了一些研究，偶然发现了[Eric S. Raymond的一篇文章] (http://www.python.org/about/success/esr/), Raymond是一个著名的、值得尊敬的黑客。其中他谈道，Python是如何成为他最喜爱的编程语言的。我也发现PyQt的绑定比Perl-QT更加成熟。所以我决定选择Python。

然后，我开始搜索Python的优秀书籍。我没能找到一本！我确实找到了一些O'Reilly的书，但是它们要么太贵，要么更像是参考手册而不是教程。于是，我免强接受了Python的随机文档。但是它过于简单和小巧。它的确给出了关于Python的妙计，但是不完整。由于我有编程经验，因此我能够对付它，但它并不适合于初学者。

在我第一次使用Python六个月后，我安装了当时最新的Red Hat 9.0 Linux，开始使用KWord。我对它很兴奋，突然冒出一个想法，用它写一些关于Python的东西。我开始写了几页，但是很快就有30页之多。然后我认眞地将其变成书的形式，使它更有用。经过几次重写，它已经达到了作为学习Python语言有用教程的水准。我将这本书作为我的贡献捐赠给开源社区。

本书开始于我在Python上的学习笔记，尽管为满足他人的口味，我做出了大量的努力，但直到现在我依然这么认为:

在开源的眞正精神中，我收到了很多热心读者的建设性意见、批评和反馈，这些帮助我改进了本书。

本书的状态

本书需要像您这样的读者的帮助，指出任何不足、难以理解或者错误之处。请写信给主要作者 或者各个译者留下您的意见和建议。

Appendix: Revision History

- 4.0
  - 19 Jan 2016
  - Switched back to Python 3
  - Switched back to Markdown, using GitBook and Spacemacs
- 3.0

  - 31 Mar 2014
  - Rewritten for Python 2 using AsciiDoc and adoc-mode.
- 2.1
  - 03 Aug 2013
  - Rewritten using Markdown and Jason Blevins' Markdown Mode
- 2.0
  - 20 Oct 2012
  - Rewritten in Pandoc format, thanks to my wife who did most of the conversion from the Mediawiki format
  - Simplifying text, removing non-essential sections such as `nonlocal` and metaclasses
- 1.90
  - 04 Sep 2008 and still in progress
  - Revival after a gap of 3.5 years!
  - Rewriting for Python 3.0
  - Rewrite using http://www.mediawiki.org[MediaWiki] (again)
- 1.20
  - 13 Jan 2005
  - Complete rewrite using Quanta+ on Fedora Core 3 with lot of corrections and updates. Many new examples. Rewrote my DocBook setup from scratch.
- 1.15
  - 28 Mar 2004
  - Minor revisions
- 1.12
  - 16 Mar 2004
  - Additions and corrections
- 1.10
  - 09 Mar 2004
  - More typo corrections, thanks to many enthusiastic and helpful readers.
- 1.00
  - 08 Mar 2004
  - After tremendous feedback and suggestions from readers, I have made significant revisions to the content along with typo corrections.
- 0.99
  - 22 Feb 2004

  - Added a new chapter on modules. Added details about variable number of arguments in functions.
- 0.98
  - 16 Feb 2004
  - Wrote a Python script and CSS stylesheet to improve XHTML output, including a crude-yet-functional lexical analyzer for automatic VIM-like syntax highlighting of the program listings.
- 0.97
  - 13 Feb 2004
  - Another completely rewritten draft, in DocBook XML (again). Book has improved a lot - it is more coherent and readable.
- 0.93
  - 25 Jan 2004
  - Added IDLE talk and more Windows-specific stuff
- 0.92
  - 05 Jan 2004
  - Changes to few examples.
- 0.91
  - 30 Dec 2003
  - Corrected typos. Improvised many topics.
- 0.90
  - 18 Dec 2003
  - Added 2 more chapters. OpenOffice format with revisions.
- 0.60
  - 21 Nov 2003
  - Fully rewritten and expanded.
- 0.20
  - 20 Nov 2003
  - Corrected some typos and errors.
- 0.15
  - 20 Nov 2003
  - Converted to DocBook XML with XEmacs.
- 0.10
  - 14 Nov 2003
  - Initial draft using KWord.

继续阅读附录: 翻译

翻译

There are many translations of the book available in different human languages, thanks to many tireless volunteers!

If you want to help with these translations, please see the list of volunteers and languages below and decide if you want to start a new translation or help in existing translation projects.

If you plan to start a new translation, please read the Translation how-to.

Arabic

Below is the link for the Arabic version. Thanks to Ashraf Ali Khalaf for translating the book, you can read the whole book online at http://www.khaledhosny.org/byte-of-python/index.html or you can download it from sourceforge.net for more info see http://itwadi.com/byteofpython_arabi.

Azerbaijani

Jahangir Shabiyev (c.shabiev@gmail.com) has volunteered to translate the book to Azerbaijani. The translation is in progress at https://www.gitbook.com/book/jahangir-sh/piton-sancmasi

Brazilian Portuguese

There are two translations in various levels of completion and accessibility. The older translation is now missing/lost, and newer translation is incomplete.

Samuel Dias Neto (samuel.arataca@gmail.com) made the first Brazilian Portuguese translation (older translation) of this book when Python was in 2.3.5 version. This is no longer publicly accessible.

Rodrigo Amaral (rodrigoamaral@gmail.com) has volunteered to translate the book to Brazilian Portuguese, (newer translation) which still remains to be completed.

Catalan

Moises Gomez (moisesgomezgiron@gmail.com) has volunteered to translate the book to Catalan. The translation is in progress.

> Moisès Gómez - I am a developer and also a teacher of programming (normally for people without any previous experience).
>
> Some time ago I needed to learn how to program in Python, and Swaroop's work was really helpful. Clear, concise, and complete enough. Just what I needed.
>
> After this experience, I thought some other people in my country could take benefit from it too. But English language can be a barrier.
>
> So, why not try to translate it? And I did for a previous version of BoP.
>
> I my country there are two official languages. I selected the Catalan language assuming that others will translate it to the more widespread Spanish.

Chinese

Translations are available at http://woodpecker.org.cn/abyteofpython_cn/chinese/ and http://zhgdg.gitcafe.com/static/doc/byte_of_python.html.

Juan Shen (orion_val@163.com) has volunteered to translate the book to Chinese.

> I am a postgraduate at Wireless Telecommunication Graduate School, Beijing University of Technology, China PR. My current research interest is on the synchronization, channel estimation and multi-user detection of multicarrier CDMA system. Python is my major programming language for daily simulation and research job, with the help of Python Numeric, actually. I learned Python just half a year before, but as you can see, it's really easy-understanding, easy-to-use and productive. Just as what is ensured in Swaroop's book, 'It's my favorite programming language now'.
>
> 'A Byte of Python' is my tutorial to learn Python. It's clear and effective to lead you into a world of Python in the shortest time. It's not too long, but efficiently covers almost all important things in Python. I think 'A Byte of Python' should be strongly recommendable for newbies as their first Python tutorial. Just dedicate my translation to the potential millions of Python users in China.

Chinese Traditional

Fred Lin (gasolin@gmail.com) has volunteered to translate the book to Chinese Traditional.

It is available at http://code.google.com/p/zhpy/wiki/ByteOfZhpy.

An exciting feature of this translation is that it also contains the *executable chinese python sources* side by side with the original python sources.

> Fred Lin - I'm working as a network firmware engineer at Delta Network, and I'm also a contributor of TurboGears web framework.
>
> As a python evangelist (:-p), I need some material to promote python language. I found 'A Byte of Python' hit the sweet point for both newbies and experienced programmers. 'A Byte of Python' elaborates the python essentials with affordable size.
>
> The translation are originally based on simplified chinese version, and soon a lot of rewrite were made to fit the current wiki version and the quality of reading.
>
> The recent chinese traditional version also featured with executable chinese python sources, which are achieved by my new 'zhpy' (python in chinese) project (launch from Aug 07).
>
> zhpy(pronounce (Z.H.?, or zippy) build a layer upon python to translate or interact with python in chinese(Traditional or Simplified). This project is mainly aimed for education.

French

Gregory (coulix@ozforces.com.au) has volunteered to translate the book to French.

Gérard Labadie (gerard.labadie@gmail.com) has completed to translate the book to French.

German

Lutz Horn (lutz.horn@gmx.de), Bernd Hengelein (bernd.hengelein@gmail.com) and Christoph Zwerschke (cito@online.de) have volunteered to translate the book to German.

Their translation is located at http://ftp.jaist.ac.jp/pub//sourceforge/a/ab/abop-german.berlios/

Lutz Horn says:

> I'm 32 years old and have a degree of Mathematics from University of Heidelberg, Germany. Currently I'm working as a software engineer on a publicly funded project to build a web portal for all things related to computer science in Germany.The main language I use as a professional is Java, but I try to do as much as possible with Python behind the scenes. Especially text analysis and conversion is very easy with Python. I'm not very familiar with GUI toolkits, since most of my programming is about web applications, where the user interface is build using Java frameworks like Struts. Currently I try to make more use of the functional programming features of Python and of generators. After taking a short look into Ruby, I was very impressed with the use of blocks in this language. Generally I like the dynamic nature of languages like Python and Ruby since it allows me to do things not possible in more static languages like Java.I've searched for some kind of introduction to programming, suitable to teach a complete non-programmer. I've found the book 'How to Think Like a Computer Scientist: Learning with Python', and 'Dive into Python'. The first is good for beginners but to long to translate. The second is not suitable for beginners. I think 'A Byte of Python' falls nicely between these, since it is not too long, written to the point, and at the same time verbose enough to teach a newbie. Besides this, I like the simple DocBook structure, which makes translating the text a generation the output in various formats a charm.

Bernd Hengelein says:

> Lutz and me are going to do the german translation together. We just started with the intro and preface but we will keep you informed about the progress we make. Ok, now some personal things about me. I am 34 years old and playing with computers since the 1980's, when the "Commodore C64" ruled the nurseries. After studying computer science I started working as a software engineer. Currently I am working in the field of medical imaging for a major german company. Although C++ is the main language I (have to) use for my daily work, I am constantly looking for new things to learn.Last year I fell in love with Python, which is a wonderful language, both for its possibilities and its beauty. I read somewhere in the net about a guy who said that he likes python, because the code looks so beautiful. In my opinion he's absolutly right. At the time I decided to learn python, I noticed that there is very little good documentation in german available. When I came across your book the spontaneous idea of a german translation crossed my mind. Luckily, Lutz had the same idea and we can now divide the work.I am looking forward to a good cooperation!

Greek

The Greek Ubuntu Community translated the book in Greek, for use in our on-line asynchronous Python lessons that take place in our forums. Contact @savvasradevic for more information.

Indonesian

Daniel (daniel.mirror@gmail.com) is translating the book to Indonesian at http://python.or.id/moin.cgi/ByteofPython.

Wisnu Priyambodo (cibermen@gmail.com) also has volunteered to translate the book to Indonesian.

Also, Bagus Aji Santoso (baguzzzaji@gmail.com) has volunteered.

Italian (first)

Enrico Morelli (mr.mlucci@gmail.com) and Massimo Lucci (morelli@cerm.unifi.it) have volunteered to translate the book to Italian.

The Italian translation is present at http://www.gentoo.it/Programmazione/byteofpython.

> *Massimo Lucci and Enrico Morelli* - we are working at the University of Florence (Italy) - Chemistry Department. I (Massimo) as service engineer and system administrator for Nuclear Magnetic Resonance Spectrometers; Enrico as service engineer and system administrator for our CED and parallel / clustered systems. We are programming on python since about seven years, we had experience working with Linux platforms since ten years. In Italy we are responsible and administrator for www.gentoo.it web site for Gentoo/Linux distrubution and www.nmr.it (now under construction) for Nuclear Magnetic Resonance applications and Congress Organization and Managements.That's all! We are impressed by the smart language used on your Book and we think this is essential for approaching the Python to new users (we are thinking about hundred of students and researcher working on our labs).

Italian (second)

An Italian translation has been created by Calvina Bice & colleagues at http://besthcgdropswebsite.com/translate/a-byte-of-python/.

Japanese

Shunro Dozono (dozono@gmail.com) is translating the book to Japanese.

Korean

Jeongbin Park (pjb7687@gmail.com) has translated the book to Korean - https://github.com/pjb7687/byte_of_python

> I am Jeongbin Park, currently working as a Biophysics & Bioinformatics researcher in Korea.
>
> A year ago, I was looking for a good tutorial/guide for Python to introduce it to my colleagues, because using Python in such research fields is becoming inevitable due to the user base is growing more and more.
>
> But at that time only few Python books are available in Korean, so I decided to translate your ebook because it looks like one of the best guides that I have ever read!
>
> Currently, the book is almost completely translated in Korean, except some of the text in introduction chapter and the appendixes.
>
> Thank you again for writing such a good guide!

Mongolian

Ariunsanaa Tunjin (luftballons2010@gmail.com) has volunteered to translate the book to Mongolian.

Update on Nov 22, 2009 : Ariunsanaa is on the verge of completing the translation.

Norwegian (bokmål)

Eirik Vågeskar is a high school student at Sandvika videregående skole in Norway, a blogger and currently translating the book to Norwegian (bokmål).

> *Eirik Vågeskar*: I have always wanted to program, but because I speak a small language, the learning process was much harder. Most tutorials and books are written in very technical English, so most high school graduates will not even have the vocabulary to understand what the tutorial is about. When I discovered this book, all my problems were solved. "A Byte of Python" used simple non-technical language to explain a programming language that is just as simple, and these two things make learning Python fun. After reading half of the book, I decided that the book was worth translating. I hope the translation will help people who have found themself in the same situation as me (especially young people), and maybe help spread interest for the language among people with less technical knowledge.

Polish

Dominik Kozaczko (dominik@kozaczko.info) has volunteered to translate the book to Polish. Translation is in progress and it's main page is available here: Ukąś Pythona.

Update : The translation is complete and ready as of Oct 2, 2009. Thanks to Dominik, his two students and their friend for their time and effort!

> *Dominik Kozaczko* - I'm a Computer Science and Information Technology teacher.

Portuguese

Fidel Viegas (fidel.viegas@gmail.com) has volunteered to translate the book to Portuguese.

Romanian

Paul-Sebastian Manole (brokenthorn@gmail.com) has volunteered to translate this book to Romanian.

> *Paul-Sebastian Manole* - I'm a second year Computer Science student at Spiru Haret University, here in Romania. I'm more of a self-taught programmer and decided to learn a new language, Python. The web told me there was no better way to do so but read "A Byte of Python". That's how popular this book is (congratulations to the author for writing such an easy to read book). I started liking Python so I decided to help translate the latest version of Swaroop's book in Romanian. Although I could be the one with the first initiative, I'm just one volunteer so if you can help, please join me.

Russian

Vladimir Smolyar (v_2e@ukr.net) has completed a Russian translation at http://wombat.org.ua/AByteOfPython/.

Ukranian

Averkiev Andrey (averkiyev@ukr.net) has volunteered to translate the book to Russian, and perhaps Ukranian (time permitting).

Serbian

"BugSpice" (amortizerka@gmail.com) has completed a Serbian translation:

> This download link is no longer accessible.

More details at http://forum.ubuntu-rs.org/Thread-zagrljaj-pitona.

Slovak

Albertio Ward (albertioward@gmail.com) has translated the book to Slovak at http://www.fatcow.com/edu/python-swaroopch-sl/ :

> We are a non-profit organization called "Translation for education". We represent a group of people, mainly students and professors, of the Slavonic University. Here are students from different departments: linguistics, chemistry, biology, etc. We try to find interesting publications on the Internet that can be relevant for us and our university colleagues. Sometimes we find articles by ourselves; other times our professors help us choose the material for translation. After obtaining permission from authors we translate articles and post them in our blog which is available and accessible to our colleagues and friends. These translated publications often help students in their daily study routine.

Spanish

Alfonso de la Guarda Reyes (alfonsodg@ictechperu.net), Gustavo Echeverria (gustavo.echeverria@gmail.com), David Crespo Arroyo (davidcrespoarroyo@hotmail.com) and Cristian Bermudez Serna (crisbermud@hotmail.com) have volunteered to translate the book to Spanish.

Gustavo Echeverria says:

> I work as a software engineer in Argentina. I use mostly C# and .Net technologies at work but strictly Python or Ruby in my personal projects. I knew Python many years ago and I got stuck inmediately. Not so long after knowing Python I discovered this book and it helped me to learn the language. Then I volunteered to translate the book to Spanish. Now, after receiving some requests, I've begun to translate "A Byte of Python" with the help of Maximiliano Soler.

Cristian Bermudez Serna says:

> I am student of Telecommunications engineering at the University of Antioquia (Colombia). Months ago, i started to learn Python and found this wonderful book, so i volunteered to get the Spanish translation.

Swedish

Mikael Jacobsson (leochingkwake@gmail.com) has volunteered to translate the book to Swedish.

Turkish

Türker SEZER (tsezer@btturk.net) and Bugra Cakir (bugracakir@gmail.com) have volunteered to translate the book to Turkish. "Where is Turkish version? Bitse de okusak."

继续阅读附录：参与翻译工作

参与翻译工作

1. The full source of the book is available from .
2. Please fork the repository.
3. Then, fetch the repository to your computer. You need to know how to use Git to do that.
4. Read the GitBook documentation, esp. the Markdown section.
5. Start editing the `.md` files to translate to your local language.
6. Sign up on GitBook.com, create a book and you can see a beautifully rendered website, with links to download PDF, EPUB, etc.

继续阅读反馈

反馈

The book needs the help of its readers such as yourselves to point out any parts of the book which are not good, not comprehensible or are simply wrong. Please write to the main author or the respective translators with your comments and suggestions.

Django Step By Step

目 录

继续阅读第一讲

Django Step by Step (一)

1 开篇

Django 是新近出来的 Rails 方式的 web 开发框架。在接触 Django 之前我接触过其它几种 Python 下的 web framework, 但感觉 Karrigell 是最容易上手的。不过 Django 从我个人的感觉上来看，它的功能更强大，社区也很活跃，高手众多，发展也是极为迅速。我个人很看好。但在学习的过程中感觉与 Karrigell 的开发体验差距比较大。那么我想，Karrigell 与 Django 的开发体验到底差别在哪里，为什么 Django 给我的感觉还不是那么清晰直观呢？

我想可能是因为 Django 的教程过于想把它的特色展示给大家，因此，对于初学者来说一下子接触的东西太多，反倒让大家很难理解。于是我想从最最简单的例子做起，并且记录下来，并且将其形成一个教程。

2 Karrigell的入门体验

不知道大家是否了解 Karrigell ，它是一个优秀的 web framework 框架，在发现它之后我写了不少关于代码分析的文章。因为它开发方式非常灵活，特别是方便，对它的印象也非常好。只不过开发 web 不是我的主业，因此实践得少。不过 Zoom.Quiet 在这方面已经有所建树，大家可以去 wiki 上学习他写的“问卷调查生成系统”的快速体验教程 。这是一个非常详细的过程，而且还有图。

那么为什么 Karrigell 开发让人感觉到方便呢？我想来想去可能有这些原因：

现在的开发，特别是 Python 的开发，我们都喜欢而且习惯边学边做，从小做起，一边做一边看效果。因此从小入手，步步有体验，这就是 Python 开发的特点。我们不会一上来就写出非常大的东西，而是写一点，运行一下，调试一下，再写一点，运行一下，调试一下，慢慢地积少成多。这种方式非常典型，也更为大多数人习惯。而 Karrigell 则基本上就是这样的。安装完 Karrigell ，然后就可以运行了。不用写程序，写个简单的 html 页面，直接放在它的webapps目录下(这是2.2版，如果你没有修改karrigell.ini的root参数的缺省目录)，在浏览器就可以看了。就这么简单。写程序也简单呀，写个hello.py，里面就是:

```
print "Hello, Karrigell!"
```

这就是最简单的web体验。从这里入手后，你就可以一点点地开始学习其它的web知识了。积少成多。 Karrigell 眞是就是这样。

3 Django的入门体验

但 Django 呢？如果说最简单的web体验 Hello, Django! 如何写呢？决不会象 Karrigell 那样简单，只从它提供的教程来看，你无法在安装后非常 Easy 地写出一个 Hello, Django! 的例子，因为有一系列的安装和准备工作要做。那么下面我把我所尝试写最简单的 Hello, Django! 的例子写出来。

> 请注意，我测试时是在 Windows环境下进行的。

3.1 安装

```
C:\>pip install django
```

3.2 生成项目目录

因为 Karrigell 可以直接开发，因此放在哪里都可以。而 Django 是一个框架，它有特殊的配置要求，因此一般不需要手工创建目录之类的工作， Django 提供了 django-admin.py 可以做这件事。

```
C:\>django-admin startproject newtest
```

这样就在当前目录下创建了一个 newtest 目录，进去入可以看到这样的目录结构：

```
newtest/
    manage.py
    newtest/
        __init__.py
        settings.py
        urls.py
        wsgi.py
```

这个 newtest 将是我们以后工作的目录，许多讲解都是基于这个目录的。

> 最外层的newtest/目录包括了项目的全部文件，这个目录名可以随意修改。
>
> manage.py： 提供简单化的 django-admin.py 命令，特别是可以自动进行 DJANGO_SETTINGS_MODULES 和 PYTHONPATH 的处理，而没有这个命令，处理上面环境变量是件麻烦的事情
>
> newtest/子目录是项目实际运行所依赖的Python包。目录名就是Python包名，如果需要导入子目录的模块就需要使用这个名字(例如mysite.urls)。
>
> newtest/**init**.py： 表示这是一个 Python 的包
>
> newtest/settings.py： 它是django的配置文件
>
> newtest/uls.py： url映射处理文件， Karrigell 没有这种机制，它通过目录/文件/方法来自动对应，而 Django 的url映射是url对于某个模块方法的映射，目前不能自动完成
>
> newtest/wsgi.py： 如果需要将项目部署到兼容WSGI的Web服务器上，这个文件就是入口。

虽然 django-admin.py 为我们生成了许多东西， 而且这些东西在以后的开发中你都需要熟悉，但现在我们的目标是最简单的体验，就认为我们不需要知道它们都有什么用吧。

项目创建好了，那么我们可以启动服务器吗？ Django 为了开发方便，自带了一个用于开发的 web server。

3.3 启动 web server

别急呀，还没看见 Hello, Django! 在哪里呢。是的，我只是想看一看， Django 能否启动。

```
C:\newtest\>python manage.py runserver
```

一旦出现:

```
Performing system checks...

System check identified no issues (0 silenced).

You have unapplied migrations; your app may not work properly until they are applied.
Run 'python manage.py migrate' to apply them.

November 25, 2016 - 15:50:53
Django version 1.10, using settings 'mysite.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
```

说明 Django 真的启来了。在浏览器中看一下，有一个祝贺页面，说明成功了。



3.4 增加一个helloworld的app吗？

在 Django 中绝大多数应用都是以app形式存在的，但一定要加吗？其实并不需要。在 Django 中，每个app就是一个子包，真正调用时需要通过 URL Dispatch 来实现url与模块方法的映射。这是 Django 的一大特色，但也是有些麻烦的地方。不用它，你无法发布一个功能，如果在 Django 中存在一种缺省的简单映射的方式，这样我想可以大大提高 Django 的入门体验度。

因此根据 URL Dispatch 的机制，我们只要保证 Django 可以在正确的地方找到方法进行调用即可。那么我们就根本不去创建一个app了。

在 newtest子目录下创建一个文件 helloworld.py 内容为:

```
from django.http import HttpResponse

def index(request):
    return HttpResponse("Hello, Django.")
```

3.5 修改urls.py

```
from django.conf.urls import url
from django.contrib import admin
from . import helloworld

urlpatterns = [
    url(r'^admin/', admin.site.urls),
    url(r'^$', helloworld.index),
]
```

好了。保存了。上面的 r'^$' 是为了匹配空串，也就是形如: http://localhost:8000/ 。如果这时 web server 已经启动了，那么直接刷新页面就行了。

现在觉得 Django 是不是简单多了，除了创建一个项目的操作，然后可能要修改两个配置文件，其它还都简单吧。

4 结论

Django 本身的确是一种松散的框架组合，它既复杂又简单。复杂是因为如果你想使用它的自动化的、高级的功能你需要学习很多的东西，而且它的教程一上来就是以这种过于完整的例子进行展示，自然会让你觉得很麻烦。不过看了我的讲解之后，是不是觉得还是挺简单的。那么我们就先以无数据库的方式进行下去，一点点地发掘 Django 的功能特性吧。

回过头来再细想一想，之所以认为 Karrigell 简单而 Django 复杂，主要在于 Karrigell 的教程适合我们这种由浅入深，循序渐近的方式，而 Django 虽然也可以这样，但它的教程却没有做成这样，因此让我们茫然。当然到最后，在我们熟练之后我们不再会有这样的感觉，毕竟这只是入门的体验，但就是这种体验可能会吓走许多的人呢。

Django Step by Step (二)

1 引言

随着学习，我们的例子也开始复杂了，下一步我想实现一个简单的 web 加法器。界面会是这样：



很简单。通过本节的学习我们可以学习到：

```
如何处理页面表格提交的数据，并且会对 URL Dispatch 作更进一步的解释。
```

2 创建 add.py 文件

我们在newtest子目录中创建一个 add.py 文件。(由于我们还没有涉及到 Django 的模型，因此象 add.py 这样的东西叫什么呢？还是称其为 View 吧。因为在 django 中，View 是用来显示的，它代替了一般的 MVC 中的 Control 的作用，因为 Django 中不是 MVC 而是 MTV (Model Template View))

```
from django.http import HttpResponse

text = """<form method="post" action="/add/">
    <input type="text" name="a" value="%d"> + <input type="text" name="b" value="%d">
    <input type="submit" value="="> <input type="text" value="%d">
</form>"""

def index(request):
    if 'a' in request.POST:
        a = int(request.POST['a'])
        b = int(request.POST['b'])
    else:
        a = 0
        b = 0
    return HttpResponse(text % (a, b, a + b))
```

这里只有一个 index 方法。所有在 view 中的方法第一个参数都会由 Django 传入 request 对象，它就是请求数据对象，它是由 Django 自动生成。其中有 GET 和 POST 属性，分别保存两种不同的提交方式的数据，它们都可以象字典一样工作。

那么我的想法就是：

进入页面就是上面的效果，页面上有两个输入文本框，一个提交按钮，一个显示结果的文本框。在两个输入文本框中输入整数，然后点击提交("="号按钮)，将返回相同的页面，但结果文本框中将显示两数相加的和。两个输入文本框分别定义为 a 和 b 。

这里的逻辑就是：先判断 POST 数据中是否有变量 a ，如果没有则表示是第一次进入，则 a, b 初始为 0 ，然后返回页面。如果有变量 a ，则计算结果，返回页面。

其实这里面有许多可以细说的东西，那么我把它们放在后面陈述。

3 修改urls.py

```
from django.conf.urls import url
from django.contrib import admin
from . import helloworld, add

urlpatterns = [
    url(r'^admin/', admin.site.urls),
    url(r'^$', helloworld.index),
    url(r'^add/$', add.index),
]
```

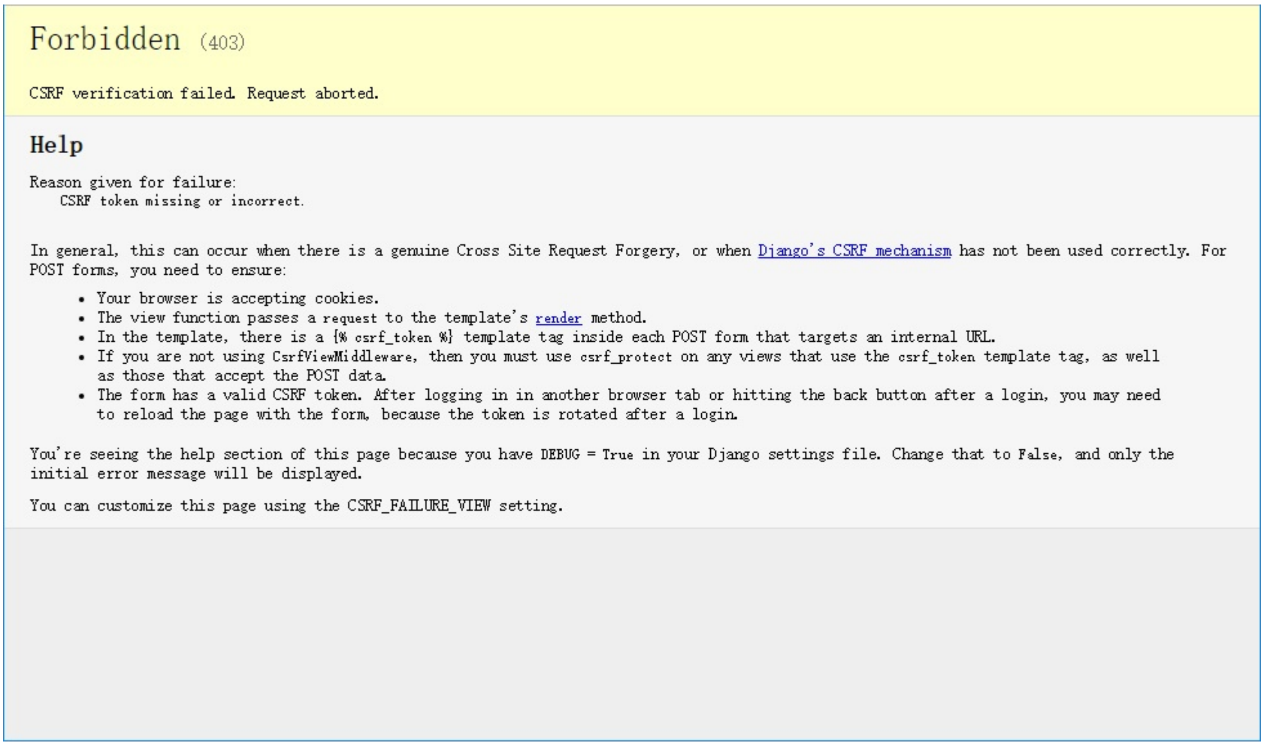
增加 add 的 url 映射。

4 启动 server

5 在浏览器测试

http://localhost:8000/add

点击提交之后，你会看到下面这个信息：



这是由于Django默认启动了防止CSRF（Cross-site request forgery：跨站请求伪造，是一种对网站的恶意利用）攻击的安全设置。本章暂不讨论这方面的内容，我们通过装饰器（decorator）关闭这个设置。

修改add.py文件：

```python
from django.http import HttpResponse
from django.views.decorators.csrf import csrf_exempt

text = """<form method="post" action="/add/">
    <input type="text" name="a" value="%d"> + <input type="text" name="b" value="%d">
    <input type="submit" value="="> <input type="text" value="%d">
</form>"""


@csrf_exempt
def index(request):
    if 'a' in request.POST:
        a = int(request.POST['a'])
        b = int(request.POST['b'])
    else:
        a = 0
        b = 0
    return HttpResponse(text % (a, b, a + b))
```

你会看到和我相似的界面，然后输入整数试一试吧。

6 补充说明

1. 在 form 中的 method="post" 。你当然可以使用 get ，但是在 Django 的设计风格中认为，使用 POST 表示要对数据进行修改，使用 GET 则只是获取，这是一个设计风格，并且不仅仅属于 Django 。如果能够养成习惯是非常好的。
2. Django 提供了 URL Dispatch 文档，专门讲解有关 url 映射的东西。其中有一部分是关于 url 的正则表达式解析的。原本我认为象 Karrigell 中一样，定义在 form 中的变量会自动映射为方法的参数，但是我错了。方法中的参数是从 url 中通过正则表达式解析出来的，或者是在 url_conf(即 urls.py 文件)中指定的。因此它与 Karrigell 一点也不一样。因此，如果你想从 POST 或 GET 数据中得到值，那么象我一样去做好了。使用 request.POST 或 request.GET 或还有一个可以“统吃”的方法 request.REQUEST ，它们是一个字典数据，使用起来也算方便。

从这里我更想了解方法中参数的使用，当然这个例子并没有，有机会再使用吧。关于正则表达式解析参数在 blog 和 rss 中用得是非常多的。

Django Step by Step (三)

1 引言

本教程只想从浅到深地将大家带入到 Django 的世界，因此都是以简单的例子出发，而且这些例子都是为了说明问题，本身并没有什么实际的意义。因此许多高级话题都无法涉及，需要大家自行看文档，做试验。

从上一例我们看到，表格的生成是直接在 index() 函数中返回的 HTML 代码，这种混合方式对于大型开发非常不好，下面我们就学习模板的使用。 Django 自带有模板系统，但你可以不使用它，只要在 return 前使用自己喜欢的模板系统进行处理，然后返回即可。但 Django 自带的模板系统有很多特点，我不做过多的说明。我只是想使用它。

现在我的问题就是：

> 我有一个通讯录数据，我想使用一个表格来显示。

为了方便，我们不需要使用数据库，因此我把它存在 view 文件中。

2 创建 list.py

```
from django.shortcuts import render_to_response

address = [
    {'name':'张三', 'address':'地址一'},
    {'name':'李四', 'address':'地址二'}
    ]

def index(request):
    return render_to_response('list.html', {'address': address})
```

这里使用了一个新方法是 render_to_response ，它可以直接调用模板并返回生成好的文本，直接返回它即可。它接收两个参数，第一个是模板的名字。

第二个参数是一个字典，这里只有一个 Key ，名字是 address ，它的值是一个字典的列表。只要注意模板所接收的就是这样的字典和包含字典的列表就行了。

3 在 newtest 中创建 templates 目录

用来存放模板文件

4 修改 settings.py

修改 `INSTALLED_APPS` 的内容，增加 `'newtest',` ，最后的设置为：

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'newtest',
]
```

Django会自动搜索newtest/templates目录下的模板文件。

5 创建 templates/list.html

```
<h2>通讯录</h2>
<table border="1">
  <tr><th>姓名</th><th>地址</th></tr>
  {% for user in address %}
  <tr>
  <td>{{ user.name }}</td>
  <td>{{ user.address }}</td>
  </tr>
{% endfor %}
</table>
```

很简单，就是生成了一个两列的表格。在 Django 模板中 `{{}}` 表示引用一个变量， `{%%}` 表示代码调用。在变量引用中， Django 还支持对变量属性的访问，同时它还有一定的策略，详细的建议查看 The Django template language 文档。这里我也使用了汉字，因此它也需要使用 utf-8 编码。

这里使用 for .. in 的模板 Tag 处理。因此 address 需要是一个集合。在我们的 View 代码中，address 为一个 list 值。每个 list 又是一个字典。因此 `{{ user.name }}` 和 `{{ user.address }}` 就是将这个字典中的元素取出来。

6 修改 urls.py

```
from django.conf.urls import url
from django.contrib import admin
from . import helloworld, add, list

urlpatterns = [
    url(r'^admin/', admin.site.urls),
    url(r'^$', helloworld.index),
    url(r'^add/$', add.index),
    url(r'^list/$', list.index),
]
```

增加了 list 的 url 映射。

7 启动 server

效果如如这样：

通讯录

姓名	地址
张三	地址一
李四	地址二

Django Step by Step (四)

1 引言

经过前几节的学习，我想大家应该比较熟悉 Django 的大致开发流程：

- 增加 view 方法
- 增加模板
- 修改 urls.py

就是这样。剩下的就是挖掘 Django 提供的其它的能力。在我们还没有进入模型(model)之前还是再看一看外围的东西，再更进一步体验 Django 吧。

在 Django 中我看到了一个生成 csv 格式的文档(Outputting CSV dynamically)，非常好，它没有数据库，正好用来做演示。

更进一步，现在我的需求就是提供 excel 格式文件的下载。

我们会在原来 list(表格) 例子基础上进行演示，步骤就是上面的流程。

2 修改 templates/list.html

在文件最后增加:

```
<p><a href="/xls/address/">Excel格式下载</a></p>
```

它将显示为一个链接，它所指向的链接将用来生成 Excel 文件。

3 在newtest下增加 xls_test.py

```
from django.http import HttpResponse
from django.template import loader, Context

address = [
    ('张三', '地址一'),
    ('李四', '地址二')
    ]

def output(request, filename):
    response = HttpResponse(content_type='application/ms-excel')
    response['Content-Disposition'] = 'attachment; filename=%s.xls' % filename

    t = loader.get_template('xls.html')
    c = Context({
        'data': address,
    })
    response.write(t.render(c))
    return response
```

这里使用的东西多了一些。这里没有 `render_to_response` 了，而是演示了一个完整的从头进行模板解析的处理过程。为什么需要这样，因为我们需要修改 `response` 对象的值，而 `render_to_response` 封装了它使得我们无法修改。从这里我们也可以看到，在调用一个方法时， Django 会传入一个 `request` 对象，在返回时，你需要将内容写入 `response` ，必要时修改它的某些属性。更详细的建议你参考 django 所带的 request_response 文档，里面详细描述了两个对象的内容，并且还可以在交互环境下进行测试，学习非常方便。

这里 `address` 不再是字典的列表，而是 tuple 的列表。让人高兴的是， Django 的模板除了可以处理字典，还可以处理序列，而且可以处理序列中的元素。一会在模板定义中我们会看到。

这里 `output()` 是我们希望 Django 调用的方法，不再是 `index()` 了。(不能老是一样的呀。)而且它与前面的 `index()` 不同，它带了一个参数。这里主要是想演示 url 的参数解析。因此你要注意，这个参数一定是放在 url 上的。它表示输出文件名。

```
response = HttpResponse(mimetype='application/ms-excel')
response['Content-Disposition'] = 'attachment; filename=%s.xls' % filename
```

这两行是用来处理输出类型和附件的，以前我也没有用过，这回也学到了。它表明返回的是一个Excel格式的文件。

```
t = loader.get_template('xls.html')
c = {
    'data': address,
}
response.write(t.render(c))
```

这几行就是最原始的模板使用方法。先通过 `loader` 来找到需要的模板，然后生成一个 template 对象，再生成一个 `Context` 对象，它就是一个字典集。然后 `t.render(c)` 这个用来对模板和提供的变量进行合并处理，生成最终的结果。最后调用 `response.write()` 将内容写入。

4 增加 templates/xls.html

```
<!DOCTYPE html>
<html lang="zh-cn">
  <head>
    <meta charset="utf-8">
  </head>
  <body>
      <table>
        {% for row in data %}
        <tr>
            <td>{{ row.0|addslashes}}</td>
            <td>{{ row.1|addslashes}}</td>
        </tr>
        {% endfor %}
  </body>
</html>
```

使用了一个 for 循环。这里 `data` 与上面的 `Context` 的 `data` 相对应。因为 `data` 是一个列表，它的每行是一个 tuple ，因此 `row.0`, `row.1` 就是取 tuple 的第一个和第二个元素。 `|` 是一个过滤符，它表示将前一个的处理结果作为输入传入下一个处理。因此 Django 的模板很强大，使用起来也非常直观和方便。 `addslashes` 是 Django 模板内置的过滤 Tag ，它用来将结果中的特殊字符加上反斜线。

同时我们注意到，每个 `{{}}` 前后都有一个双引号，这样就保证每个字符串使用双引号引起来。然后在第一个与第二个元素之间还使用了逗号分隔。最后 `endfor` 在下一行，表示上面每行模板后有一个回车。

Django 还允许你自定义 Tag ，在 The Django template language: For Python programmers 文档中有描述，其实是很简单的。

5 修改 urls.py

```
from django.conf.urls import url
from django.contrib import admin
from . import helloworld, add, list, xls_test

urlpatterns = [
    url(r'^admin/', admin.site.urls),
    url(r'^$', helloworld.index),
    url(r'^add/$', add.index),
    url(r'^list/$', list.index),
    url(r'^xls/(?P<filename>\w+)/$', xls_test.output),
]
```

增加了 xls 的 url 映射。

上面的正则表达式有些复杂了，因为有参数的处理在里面。 `(?P<filename>\w+)` 这是一个将解析结果起名为 `filename` 的正则表达式，它完全符合 Python 正则表达式的用法。在最新的 Django 中，还可以简化一下： `(\w+)` 。但这样需要你的参数是按顺序传入的，在一个方法有多个参数时一定要注意顺序。

还记得吗？我们的链接是写成 `/xls/address/` ，因此上面实际上会变成对 `xls_test.output(filename='address')` 的调用。

6 启动 server

看一下结果吧。点击链接，浏览器会提示你保存文件的。

很简单吧。但这里面的内容其实也不少，而且许多地方都有很大的扩展空间。

Django Step by Step (五)

1 引言

其实本教程以展示基本概念为已任，对于一些高级的话题我也在不停地学习中，希望能有所展示。让我们一起学习吧。

在了解了基本的 Django 开发的过程及 Django 的一些基本特性之后，越来越多的东西在等着我们。现在我们就学习一下 session 吧。 session 可以翻译为“会话”，做过web的可能都知道。它就是为了实现页面间的数据交换而产生的东西，一般有一个 session_id ，它会保存在浏览器的 cookie 中，因此如果你的浏览器禁止了 cookie ，下面的试验是做不了的。

在 Django 中的 session 也非常简单，它就存在于 `request` 对象的 `session` 属性中。你可以把它看成一个字典就可以了。

下面我们做一个非常简单的功能：首先当用户进入某个页面，这个页面会显示一个登录页面，上面有一个文本框用来输入用户名，还有一个提交按钮用来提交数据。当用户输入用户名，然后点提交，则显示显示用户已经登录，并且打印出用户的姓名来，同时还提供一个“注销”按钮。然后如果用户再次进入这个页面，则显示同登录成功后的页面。如果点击注销则重新进入未登录的页面。

2 在newtest下创建 login.py

```
from django.http import HttpResponseRedirect
from django.shortcuts import render_to_response
from django.views.decorators.csrf import csrf_exempt

@csrf_exempt
def login(request):
    username = request.POST.get('username', None)
    if username:
        request.session['username'] = username
    username = request.session.get('username', None)
    if username:
        return render_to_response('login.html', {'username':username})
    else:
        return render_to_response('login.html')

@csrf_exempt
def logout(request):
    try:
        del request.session['username']
    except KeyError:
        pass
    return HttpResponseRedirect("/login/")
```

有些复杂了吗？没关系，让我解释一下。这里有两个方法： `login()` 和 `logout()` 。 `login()` 用来提供初始页面、处理提供数据和判断用户是否登录。而 `logout()` 只是用来从 session 中删除用户名，同时将页面重定向到 login 画面。这里我仍然使用了模板，并且根据传入不同的字典来控制模板的生成。是的，因为 Django 的模块支持条件判断，所以可以做到。

在 `login()` 中的判断逻辑是：

- 先从 POST 中取 `username` (这样 `username` 需要由模板的 form 来提供)，如果存在则加入到 session 中去。加入 session 很简单，就是一个字典的 Key 赋值。
- 然后再从 session 中取 `username` ，有两种可能：一种是上一步实现的。还有一种可能是直接从以前的 session 中取出来的，它不是新产生的。而这里并没有细分这两种情况。因此这个判断其实对应两种页面请求的处理：一种是提交了用户姓名，而另一种则是处理完用户提交姓名之后，用户再次进入的情况。而用户再次进入时，由于我们在前面已经将他的名字保存在 session 里面了，因此可以直接取出来。如果 session 中存在，则表示用户已经登录过，则输出 `login.html` 模板，同时传入了 `username` 字典值。而如果 session 中不存在，说明用户从来没有登录过，则输出 `login.html` 模板，这次不带值。

因此对于同一个 `login.htm` 模板传入的不同值，后面我们会看到模板是如何区分的。

在 `logout()` 中很简单。先试着删除 session ，然后重定向页面到 login 页面。这里使用了 `HttpResponseRedirect` 方法，它是从以前我们看到的 `HttpResponse` 派生来的子类。更多的派生子类和关于 response 的内容要参考 Request and response objects 文档。

3 创建 templates/login.html

```
{% if not username %}
<form method="post" action="/login/">
    用户名：<input type="text" name="username" value=""><br/>
    <input type="submit" value="登录">
</form>
{% else %}
你已经登录了！{{ username }}<br/>
<form method="post" action="/logout/">
    <input type="submit" value="注销">
</form>
{% endif %}
```

整个是一个 `if` 语句。在 Django 模板中的 `if` 可以象 Python 一样使用，如使用 `not` ， `and` ， `or` 。象 `if not username` 表示什么呢？它表示如果 username 不存在，或为空，或是假值等等。而此时我们利用了 `username` 不存在这种判断。

上面的逻辑表示，如果 `username` 不存在，则显示一个表单，显示用户名输入文本框。如果存在，则显示已经登录信息，同时显示用户名和注销按钮。而这个注销铵钮对应于 `logout()` 方法。

4 修改 urls.py

```
from django.conf.urls import url
from django.contrib import admin
from . import helloworld, add, list, xls_test, login

urlpatterns = [
    url(r'^admin/', admin.site.urls),
    url(r'^$', helloworld.index),
    url(r'^add/$', add.index),
    url(r'^list/$', list.index),
    url(r'^xls/(?P<filename>\w+)/$', xls_test.output),
    url(r'^login/$', login.login),
    url(r'^logout/$', login.logout),
]
```

增加了 login 和 logout 两个url映射。

5 启动 server 运行

但我要说，你一定会报错。而且我的也在报错。为什么，因为从这一刻起，我们就要进入有数据库的环境了。因为在 django 中 session 是存放在数据库中的。所以在这里要进行数据库的初始化了。

6 查看 settings.py

我们在创建newtest工程的时候，Django已经为我们默认生成了数据库处理的配置:

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': os.path.join(BASE_DIR, 'db.sqlite3'),
    }
}
```

这里使用的是 sqlite3 。在使用数据库时，你同时需要自已去安装相应的数据库处理模块。

7 初始化数据库

配置不需要做什么修改，下一步我们进行数据库的初始化工作，包括建库、建表等:

```
python manage.py migrate
```

8 启动 server

这次再进入试吧

(http://localhost:8000/login/)

从此我们要进入数据库的世界了，当然目前还没有用到，而 Django 提供的许多自动化的高级功能都是需要数据库支持的。

Django Step by Step (六)

1 引言

以后的例子可能会越来越复杂，没办法因为我们用的东西越来越复杂，同时我们的能力也在增长。

下面我们按照 TurboGears 的 Wiki in 20 Minutes 的例子仿照一个，我们要用 Django 来做 wiki。我不会按 TurboGears 的操作去做，只是实现一个我认为的最简单的 wiki。

现在我的要求是：

> 做一个简单的wiki，要可以修改当前页面，即在页面下面提供一个编辑的按钮。然后还要识别页面中的两个开头大写的单词为页面切换点，可以进入一个已经生成好的页面，或提示创建一个新页面。

下面我们将开始创建 Django 中的 app 了。

先说一下。如果你看过官方版的教程，它就是讲述了一个 Poll 的 app 的生成过程。那么一个 app 就是一个功能的集合，它有自已的 model ，view 和相应的模板，还可以带自已的 urls.py 。那么它也是一个独立的目录，这样一个 app 就可以独立地进行安装，你可以把它安装到其它的 Django 服务器中去。因此采用 app 的组织形式非常有意义。而且 `adango-admin.py` 也提供了一个针对 app 的命令，一会我们就会看到。而且 Django 提供一些自动功能也完全是针对于 app 这种结构的。Model, Template, View 就合成了 MTV 这几个字母。 Model 是用来针对数据库，同时它可以用来自动生成管理界面， View 在前面我们一直都用它，用来处理请求和响应的相当于MVC框架中的 Controller 的作用， Template 用来生成界面。

2 创建 wiki app

```
python manage.py startapp wiki
```

这样在 wiki 子目录下有以下文件:

- **init**.py 表示 wiki 目录是一个包。
- views.py 用来放它的 view 的代码。
- models.py 用来放 model 代码。
- apps.py 用来放配置代码

- admin.py 用来配置当前的wiki如何使用Django Admin功能
- tests.py 用来放测试代码
- migrations目录 用来放每一次数据库变化后需要对数据库做的变化

3 编辑 wiki/models.py

```
from django.db import models

# Create your models here.
class Wiki(models.Model):
    pagename = models.CharField(max_length=20, unique=True)
    content = models.TextField()
```

每个 model 其实在 Django 中就是一个表，你将用它来保存数据。在实际的应用中，一般都要与数据库打交道，如果你不想用数据库，那么原因可能就是操作数据库麻烦，创建数据库环境也麻烦。但通过 Django 的 model 处理，它是一种 ORM (Object Relation Mapping, 对象与关系的映射)，可以屏蔽掉底层数据库的细节，同时提供以对象的形式来处理数据。非常方便。而且 Django 的 model 层支持多种数据库，如果你改变数据库也不是什么问题，这也为以后的数据库迁移带来好处。总之，好处多多，大家多多体会吧。

Wiki 是 model 的名字，它需要从 models.Model 派生而来。它定义了两个字段，一个是字段是 pagename ，用来保存 wiki 页面的名字，它有两个参数，一个是最大长度(不过从这点上不如 SQLAlchemy 方便, SQLAlchemy并不需要长度，它会根据有无长度自动转为 TEXT 类型)，目前 CharField 需要这个参数；另一个是 unique 表示这个字段不能有重复值。还有一个字段是 content ，用来保存 wiki 页面的内容，它是一个 TextField 类型，它不需要最大长度。

现在不太了解 model 没有关系，关键是看整个生成过程。

一旦你定义好了 model ，在运行时， Django 会自动地为这个 model 增加许多数据操作的方法。关于 model 和 数据库操作API的详细内容参见 Model reference 和 Database API reference 的文档。

4 修改 settings.py, 安装 app

虽然我们的其它工作没有做完，但我还是想先安装一下 app 吧。每个一 app 都需要安装一下。安装一般有两步：

4.1 修改settings.py

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'newtest',
    'wiki.apps.WikiConfig',
]
```

这个在文件的最后，django开头的是缺省定义的。给出指定 wiki 包的引用名来。这一步是为了以后方便地导入所必须的。因为我们的目录都是包的形式，因此这里就是与目录相对应的。

4.2 执行(在newtest目录下)

```
python manage.py makemigrations
python manage.py migrate
```

如果没有报错就是成功了。这一步 Django 将根据 model 的信息在数据库中创建相应的表。表就是这样创建出来的。

5 在命令行下加入首页(FrontPage)

我们假设首页的名字为 FrontPage ，并且我们将在命令行下增加它，让我们熟悉一下命令行的使用

进入 newtest 目录，然后:

```
python manage.py shell
```

进入 python

```
>>> from wiki.models import Wiki
>>> page = Wiki(pagename='FrontPage', content='Welcome to Easy Wiki')
>>> page.save()
>>> Wiki.objects.all()
[<Wiki object>]
>>> p = Wiki.objects.all()[0]
>>> p.pagename
'FrontPage'
>>> p.content
'Welcome to Easy Wiki'
```

在 Django 中，对于数据库的记录有两种操纵方式，一种是集合方式，一种是对象方式。集合方式相当于表级操作，可以使用 model.objects 来处理。 objects 对象有一些集合方式的操作，如 all() 会返回全部记录， filter() 会根据条件返回部分记录。而象插入新记录则需要使用记录方式来操作，些时要直接使用 model 类。

6 修改 wiki/views.py

```
from .models import Wiki
from django.template import loader, Context
from django.http import HttpResponse, HttpResponseRedirect
from django.shortcuts import render_to_response
from django.views.decorators.csrf import csrf_exempt

def index(request, pagename=""):
    """显示正常页面，对页面的文字做特殊的链接处理"""
    if pagename:
        #查找是否已经存在页面
#         pages = Wiki.objects.get_list(pagename__exact=pagename)
        pages = Wiki.objects.filter(pagename=pagename)
        if pages:
            #存在则调用页面模板进行显示
            return process('wiki/page.html', pages[0])
        else:
            #不存在则进入编辑画面
            return render_to_response('wiki/edit.html', {'pagename':pagename})

    else:
#         page = Wiki.objects.get_object(pagename__exact='FrontPage')
        page = Wiki.objects.get(pagename='FrontPage')
        return process('wiki/page.html', page)

@csrf_exempt
def edit(request, pagename):
    """显示编辑存在页面"""
#     page = Wiki.objects.get_object(pagename__exact=pagename)
    page = Wiki.objects.get(pagename=pagename)
    return render_to_response('wiki/edit.html', {'pagename':pagename, 'content':page.c
```

```
ontent})

@csrf_exempt
def save(request, pagename):
    """保存页面内容，老页面进行内容替换，新页面生成新记录"""
    content = request.POST['content']
#    pages = Wiki.objects.get_list(pagename__exact=pagename)
    pages = Wiki.objects.filter(pagename=pagename)
    if pages:
        pages[0].content = content
        pages[0].save()
    else:
        page = Wiki(pagename=pagename, content=content)
        page.save()
    return HttpResponseRedirect("/%s" % pagename)

import re

r = re.compile(r'\b(([A-Z]+[a-z]+){2,})\b')
def process(template, page):
    """处理页面链接，并且将回车符转为<br>"""
    t = loader.get_template(template)
    content = r.sub(r'<a href="/\1">\1</a>', page.content)
    content = re.sub(r'[\n\r]+', '<br>', content)
    c = {'pagename':page.pagename, 'content':content}
    return HttpResponse(t.render(c))
```

代码有些长，有些地方已经有说明和注释了。简单说一下：

- `index()` 用来显示一个 wiki 页面。它需要一个参数就是页面的名称。如果在数据库中找得到，则调用 `process()` 方法(`process()` 方法是一个自定义方法，主要用来对页面的文本进行处理，比如查找是否有满足 wiki 命名规则的单词，如果有则替换成链接。再有就是将回车转为 `<br>`)。如果没有找到，则直接调用编辑模板显示一个编程页面。当然，这个页面的内容是空的。只是它的页面名字就是 `pagename` 。如果 `pagename` 为空，则进入 `FrontPage` 页面。 `Wiki.objects` 对象有 `filter()` 方法和 `get()` 方法，一个返回一个结果集，一个返回指定的对象。这里为什么使用 `filter()` 呢，因为一旦指定文件不存在，它并不是返回一个 `None` 对象，而是抛出异常，而我没有使用异常的处理方式。通过 `filter()` 如果存在则结果中应有一个元素，如果不存在则应该是一个 `[]` 。这样就知道是否有返回了。

Note

> filter() 中使用的参数与一般的 db-api 是一样的，但如果是比较相等，可以为：pagename__exact=pagename 也可以简化为 pagename=pagename 。

Note

> 在 Django 中，一些字段的比较操作比较特殊，它是在字段名后加 __ 然后是比较条件。这样看上去就是一个字符串。具体的参见 The Database API 。

Note

> 回车转换的工作其实可以在模板中使用 filter 来完成。

- 在上一章我们将所有的模板都放在了 `newtest/templates` 目录下，从本章开始，为了区分方便，我们会针对每一个app创建 `templates/app` 的子目录，将模板文件(edit.html)放在app目录下统一管理。由于Django针对TEMPLATES的默认的设置有 `'APP_DIRS': True` ，会自动到每一个app的 `templates` 目录下寻找模板文件。

因为我们在设计 model 时已经设置了 pagename 必须是唯一的，因此一旦 filter() 有返回值，那它只能有一个元素，而 pages[0] 就是我们想要的对象。

- page = wikis.get(pagename='FrontPage')

  是表示取出 pagename 为 FrontPage 的页面。你可能要说，为什么没有异常保护，是的，这也就是为什么我们要在前面先要插条记录在里面的原因。这样就不会出错了。再加上我要做的 wiki 不提供删除功能，因此不用担心会出现异常。

- `edit()` 用来显示一个编辑页面，它直接取出一个页面对象，然后调用 wiki/edit.html 模板进行显示。也许你还是要问，为什么不考虑异常，因为这里不会出现。为什么？因为 edit() 只用在已经存在的页面上，它将用于存在页面的修改。而对于不存在的页面是在 index() 中直接调用模板来处理，并没有直接使用这个 edit() 来处理。也许你认为这样可能不好，但由于在 edit() 要重新检索数据库，而在 index() 已经检索过一次了，没有必要再次检索，因此象我这样处理也没什么不好，效率可能要高一些。当然这只是个人意见。

- save() 用来在编辑页面时用来保存内容的。它先检查页面是否在数据库中存在，如果不存在则创建一个新的对象，并且保存。注意，在 Django 中，对对象处理之后只有调用它的 save() 方法才可以真正保存到数据库中去。如果页面已经存在，则更新页面的内容。处理之后再重定向到 index() 去显示这个页面。

7 在 wiki 中创建 templates 子目录

8 编辑 wiki/templates/wiki/page.html

```
<h2>{{ pagename }}</h2>
<p>{{ content }}</p>
<hr/>
<p>
<form method="POST" action="/wiki/{{ pagename }}/edit/">
<input type="submit" value="编辑">
</form></p>
```

它用来显示页面，同时提供一个“编辑”按钮。当点击这个按钮时将调用 view 中的 edit() 方法。

9 编辑 wiki/templates/wiki/edit.html

```
<h2>编辑:{{ pagename }}</h2>
<form method="POST" action="/wiki/{{pagename}}/save/">
<textarea name="content" rows="10" cols="50">{{ content }}</textarea><br/>
<input type="submit" value="保存">
</form>
```

它用来显示一个编辑页面，同时提供“保存”按钮。点击了保存按钮之后，会调用 view 中的 save() 方法。

10 修改 urls.py

```
from django.conf.urls import include, url
from django.contrib import admin
from . import helloworld, add, list, xls_test, login

urlpatterns = [
    url(r'^admin/', admin.site.urls),
    url(r'^$', helloworld.index),
    url(r'^add/$', add.index),
    url(r'^list/$', list.index),
    url(r'^xls/(?P<filename>\w+)/$', xls_test.output),
    url(r'^login/$', login.login),
    url(r'^logout/$', login.logout),
    url(r'^wiki/', include('wiki.urls')),
]
```

在wiki目录下增加一个urls.py的文件，然后编辑内容增加了 wiki 等4个 url 映射。

```
from django.conf.urls import url

from . import views

urlpatterns = [
    url(r'^$', views.index),
    url(r'^(?P<pagename>\w+)/$', views.index),
    url(r'^(?P<pagename>\w+)/edit/$', views.edit),
    url(r'^(?P<pagename>\w+)/save/$', views.save),
]
```

这里要好好讲一讲 URL 的设计(个人所见)。

一般一个 wiki ，我们访问它的一个页面可能为：wiki/pagename。因此我设计对 index() 方法的调用的 url 为:

```
r'^wiki/(?P<pagename>\w+)/$'
```

也就是把 wiki/后面的解析出来作为 `pagename` 参数。但这样就带来一个问题，如果我想实现 `wiki/edit.html` 表示修改， `pagename` 作为一个参数通过 POST 来提交好象就不行了。因为上面的解析规则会“吃”掉这种情况。因此我采用 Zope 的表示方法：把对象的方法放在对象的后面。我可以把 `pagename` 看成为一个对象， `edit` , `save` 是它的方法，放在它的后面，也简单，也清晰。当然如果我们加强上面的正则表达式，也可以解析出 `wiki/edit.html` 的情况，但那就是你设计的问题了。这里就是我的设计。

因此 wiki/pagename 就是显示一个页面，wiki/pagename/edit 就是编辑这个页面，wiki/pagename/save 就是保存页面。而 `pagename` 解析出来后就是分别与 `index()` ， `edit()` ， `save()` 的 `pagename` 参数相对应。

下面你可以运行了。

11 启动 server 进入 (http://localhost:8000/wiki)

首先进入这个页面：



然后你点编辑，则进入FrontPage的编辑界面：



然后我们加上一个 TestPage ，它符合 wiki 的名字要求，两个首字母大写的单词连在一起。然后点击保存。



看见了吧。页面上的 TestPage 有了链接。点击它将进入：



这是 TestPage 的编辑页面。让我们输入中文，然后输入 FrontPage 。然后保存。



好了，剩下的你来玩吧。点击 FrontPage 将回到首页。

Django Step by Step (七)

1 引言

敢问路在何方，路在脚本。如果你坚持下来，一定会有收获的。

直到目前我们已经学了：

- settings.py的设置
- url dispatcher
- 模板
- session
- app
- model

其实在某些方面，使用 Django 还可以更加方便。而且我们还有许多东西没有学，一点点跟着我学吧。

我有一个通讯录，它是保存在 Excel 文件中的，我不想每次到目录下去打开它，我希望用 Django 做一个web上的简单应用，如何做呢？

2 创建 address app

```
python manage.py startapp address
```

这样就创建好了 address 相关的目录了。

3 修改 address/models.py

```
from django.db import models

# Create your models here.

class Address(models.Model):
    name = models.CharField('姓名', max_length=6, unique=True)
    gender = models.CharField('性别', choices=(('M', '男'), ('F', '女')),
        max_length=1)
    telphone = models.CharField('电话', max_length=20)
    mobile = models.CharField('手机', max_length=11)
```

这回 model 复杂多了。在上面你可以看到我定义了四个字段： name , gender , telpnone , mobile 。其中 gender 表示性别，它可以从一个 tuple 数据中进行选取。并且在后面的 radio_admin=True 表示在 admin 的管理界面中将使用 radio 按钮来处理。

Note

> Django 提供了许多的字段类型，有些字段类型从数据的取值范围来讲没有什么区别，但之所以有这种区别，是因为：Django 的数据类型不仅仅用于创建数据库，进行 ORM 处理，还用于 admin 的处理。一方面将用来对应不同的 UI 控件，另一方面提供对不同的数据类型将进行不同的数据校验的功能。

在 Django 中每个字段都可以有一个提示文本，它是第一个参数，如果没有则会使用字段名。因此我定义的每个字段为了方便都有一个对应的汉字提示文本。

因为本节主要是讲 admin 的使用。admin 是 Django 提供的一个核心 app(既然是 app 就需要安装，一会就看到了)，它可以根据你的 model 来自动生成管理界面。我为什么要用它，因为有了这个管理界面，对于通讯录的增加、删除、修改的处理界面完全可以通过 admin 来自动生成，我不用自己写。不相信吗？我们就会看到了。

那么 admin 到底可以带来些什么好处呢？它的功能很强大，不仅界面漂亮，还能对数据提供操作记录，提供搜索。特别是它是在用户权限控制之下，你都可以不用考虑安全的东西了。并且它本身就是一个非常好的学习的东西，特别是界面自动生成方面，学习它的代码可以用在我们自已的定制之中。当然，你许你用不上 admin ，它的确有一定的适应范围，不过对于大部分工作来说它可能足够了。对于那些交互性强的功能，你可能要自已实现许多东西，对于管理集中，主要以发布为主的东西，使用它可以节省你大量的时间。至于怎么使用，你要自已去权衡。但这一点对于快速实现一个 web 应用，作用非常大，这是 Django 中的一个亮点。

4 修改 settings.py

```
INSTALLED_APPS = (
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'newtest',
    'wiki.apps.WikiConfig',
    'address.apps.AddressConfig',
)
```

这里我们加入了Address的app ，我们注意到系统默认就已经添加了admin的应用，就是 django.contrib.admin 。 admin 也是一个应用，需要加入INSTALLED_APPS才可以使用，这些与标准的app的安装没有什么不同。

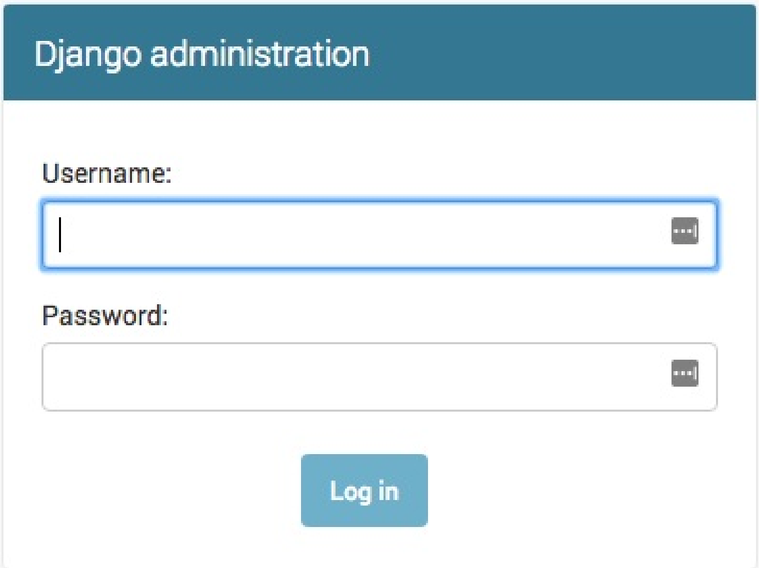
5 安装 address app

```
python manage.py makemigrations
python manage.py migrate
```

这样将在数据库中创建 address 相关的表。

6 增加超级用户

进入 (http://localhost:8000/admin)

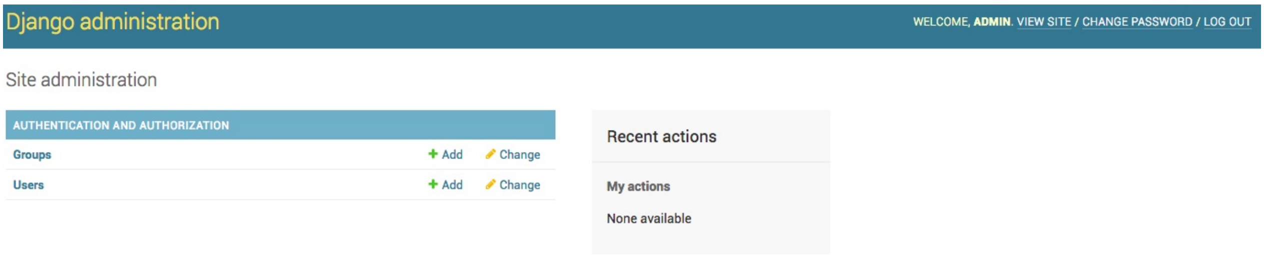


进入看一看吧。咦，要用户 。对，admin 功能是有用户权限管理的，因此一个 admin 替你完成了大量的工作：用户的管理和信息的增加、删除、修改这类功能类似，开发繁琐的东西。那么我们目前还没有一个用户，因此可以在命令下创建一个超级用户，有了这个用户，以后就可以直接在 admin 界面中去管理了。

```
python manage.py createsuperuser
```

它会让你输入用户名，邮件地址和口令。

这回再进去看一下吧。



上面已经有一些东西了，其中就有用户管理。但如何通过 admin 增加通讯录呢？别急，我们需要编辑一下address/admin.py，告诉admin应用我们的Address对象可以被admin管理。

Note

> 因此是否启用 admin 管理取决于你。只要在 address/admin.py 中增加 admin 相关的部分，我们的应用才可以在 admin 中被管理。

7 修改 address/admin.py

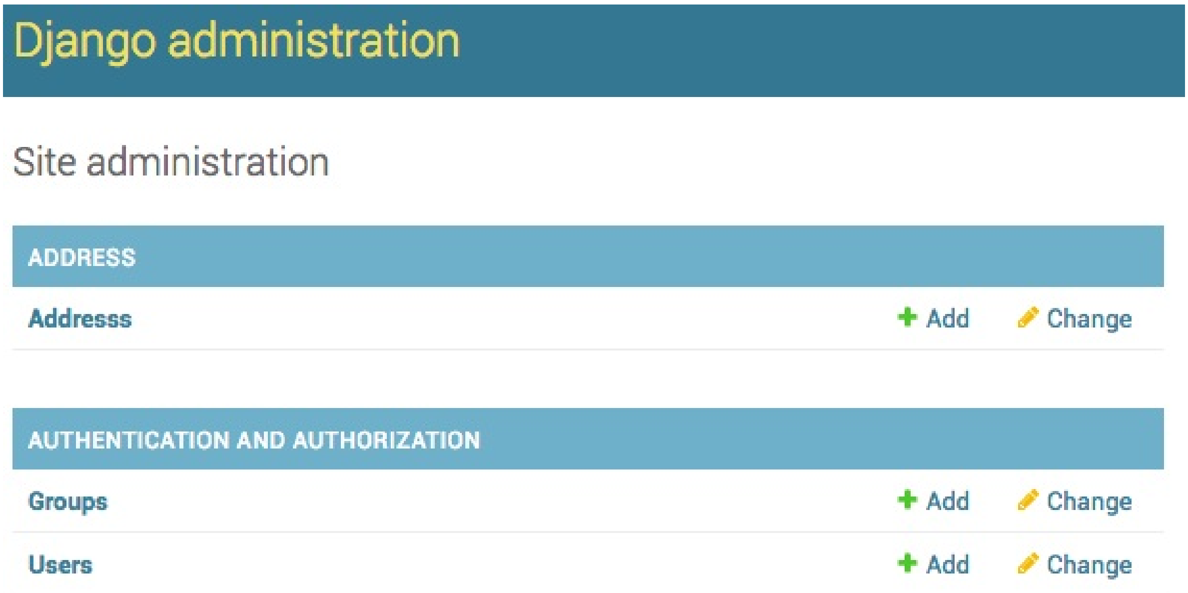
```
from django.contrib import admin

# Register your models here.

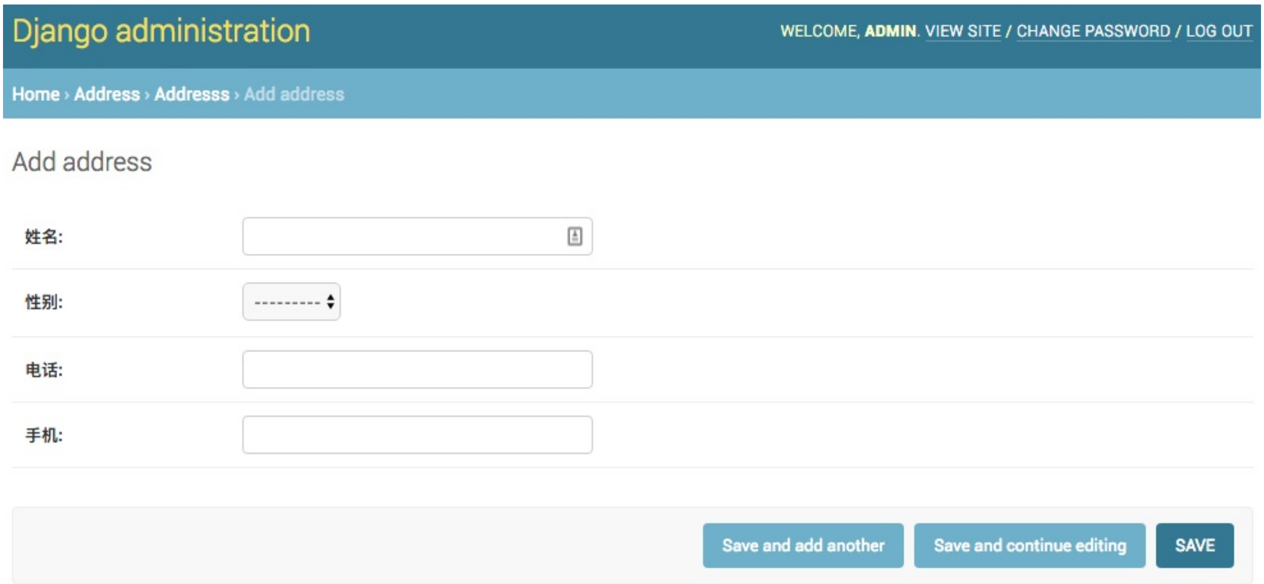
from .models import Address

admin.site.register(Address)
```

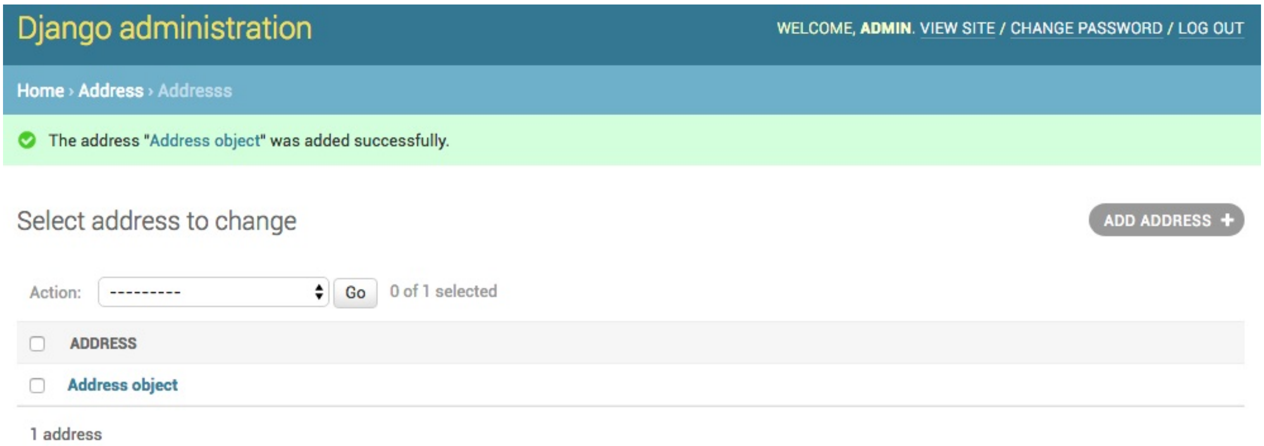
有了这个东西，你就可以在 admin 中看到 adress 这个 app 了。再到浏览器中看一下是什么样子了。



看见了吧。上面有增加和删除的按钮，先让我们点击一下增加吧。



这个自动生成的界面是不是很不错。增加一条保存起来了。不过我发现当我输入 `limodou` 时，只能输入 `limodo` 好象 `u` 输不进去。为什么？因为我把姓名按汉字算最多6个就够了，一旦我使用英文的名字可能就不够。因此这是一个问题，一会要改掉。



怎么新增的记录叫

这样看上去很别扭。为什么会这样，因为没有定义特殊的方法。下面就让我们定义一下。

8 修改 address/models.py

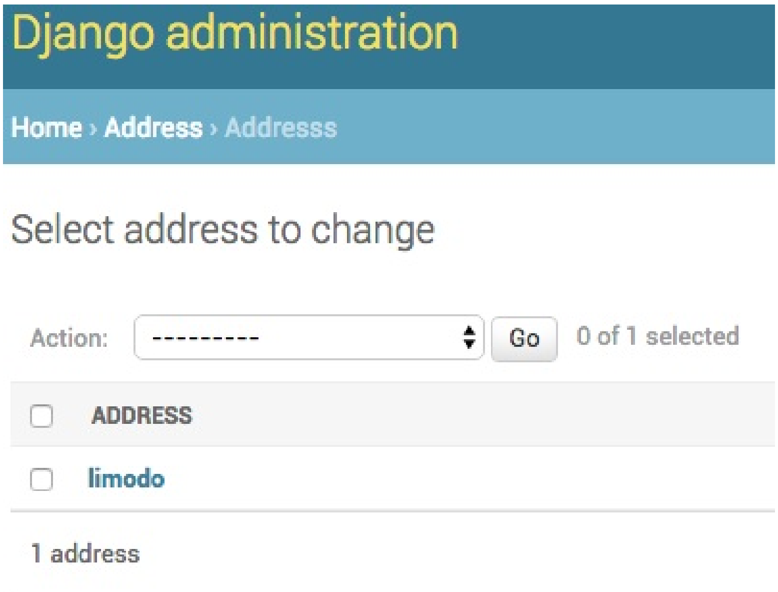
```
from django.db import models

# Create your models here.

class Address(models.Model):
    name = models.CharField('姓名', max_length=6, unique=True)
    gender = models.CharField('性别', choices=(('M', '男'), ('F', '女')),
        max_length=1)
    telphone = models.CharField('电话', max_length=20)
    mobile = models.CharField('手机', max_length=11)

    def __str__(self):
        return self.name
```

改好了，再刷新下页面。这次看见了吗？增加了一个 **str** 方法。这个方法将在显示 Address 实例的时候起作用。我们就使用某个联系人的姓名就行了。



你记得吗？Model 是与数据库中的表对应的，为什么我们改了 model 代码，不需要重新对数据库进行处理呢？因为只要不涉及到表结构的调整是不用对表进行特殊处理的。不过，我们马上要修改表结构了。

9 修改 address/models.py

姓名留短了真是不方便，另外我突然发现需要再增加一个房间字段。

```python
from django.db import models

# Create your models here.

class Address(models.Model):
    name = models.CharField('姓名', max_length=20, unique=True)
    gender = models.CharField('性别', choices=(('M', '男'), ('F', '女')),
        max_length=1)
    telphone = models.CharField('电话', max_length=20)
    mobile = models.CharField('手机', max_length=11)
    room = models.CharField('房间', max_length=10, default='')

    def __str__(self):
        return self.name
```

这回表结构要改变了，怎么做呢？

10 修改表结构

我们可以使用Django提供的数据迁移功能，在不影响原有数据的情况下，更新表结构。主要包括下面的命令：

- migrate，负责应用迁移，以及取消应用并列出其状态。
- makemigrations, 负责基于你的模型修改创建一个新的迁移
- sqlmigrate, 展示迁移的sql语句

大家一定已经注意到了，我们增加的字段 `room` 多了一个默认值空字符串，这是为了在增加一个新的字段中之后，可以使用这个默认值填充已有的记录。下面我们创建一个新的迁移

```
python manage.py makemigrations
```

然后我们就在address/migrations目录下得到了一个新的 `0002_auto_XXXXXXXX_XXXX.py` 的文件，这个文件由Django自动生成，用于迁移数据库使用，我们不用管它，使用下面的命令执行迁移：

```
python manage.py migrate
```

现在数据库中已经是新的表结构了，我们可以对数据继续进行操作。

11 进入 admin

我们可以再次进入 admin 了，增加，删除，修改数据了。

用了一会，也许你会希望：能不能有汉化版本的界面呢？答案是肯定的，而且已做好了。

12 修改 settings.py

在 `MIDDLEWARE` 部分，增加 `django.middleware.locale.LocaleMiddleware` ，代码如下：

```
MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
    'django.middleware.locale.LocaleMiddleware',
]
```

刷新下界面，是不是变成汉字了。

国际化支持在 Django 中做得是非常的出色，程序可以国际化，模板可以国际化，甚至js都可以国际化。这一点其它的类似框架都还做不到。而国际化的支持更是 RoR 的一个弱项，甚至在 Snakes and Rubies 的会议上，RoR 的作者都不想支持国际化。但 Django 却做得非常出色，目前已经有二十多种语言译文。

在增加，删除，修改都做完了，其实还剩下什么呢？显示和查询。那么实现它则需要写 view 和使用模板了。这个其实也没什么，最简单的，从数据库里查询出所有的数据，然后调用模板，通过循环一条条地显示。不错是简单。但是在做之前，先让我们想一想，这种处理是不是最常见的处理方法呢？也许我们换成其它的应用也是相似的处理。如果很多这样的处理，是不是我们需要每次都做一遍呢？有没有通用的方便的方法。答案是：有！ Django 已经为我们想到了，这就是 Generic views 所做的。它把最常见的显示列表，显示详细信息，增加，修改，删除对象这些处理都已经做好了一个通用的方法，一旦有类似的处理，可以直接使用，不用再重新开发了。但在配置上有特殊的要求。具体的可以看 Generic views 文档。

从这里我有一点想法，我认为 view 这个名称特别容易让人产生误解，为什么呢？因为 view 可以译为视图，给人一种与展示有关的什么东西。但实际上 Django 中的 view 相当于一个 Controller 的作用，它是用来收集数据，调用模板，眞正的显示是在模板中处理的。因此我倒认为使用 Controller 可能更合适，这样就称为 MTC 了。呵呵，只是个人想法。

另外， Generic views 产生的意义在于 Django 的哲学理含 DRY (Don't repeat yourself, 不要自已重复)，目的是重用，减少重复劳动。还有其它的哲学理含参见 Design philosophies 文档。

因此可以知道 view 可以极大地简化， Django 在这点上认为：每个应用的显示都可能是不同的，因此这件事需要用户来处理。但如果有最简单的封装，对于开发人员在测试时会更方便，但目前没有，因此模板我们还是要准备，而且还有特殊的要求，一会就看到了。

对于目前我这个简单的应用来说，我只需要一个简单的列表显示功能即可，好在联系人的信息并不多可以在一行显示下。因此我要使用 `django.views.generic` 模块来处理。

13 增加 address/urls.py

对，我们为 address 应用增加了自已的 urls.py。

```
from django.conf.urls import url

from . import views

urlpatterns = [
    url(r'^$', views.IndexView.as_view(), name='index'),
]
```

我们使用 `as_view` 这个generic view的方法显示默认的列表界面，可以大大的简化views.py的编码工作，现在我们的views.py代码如下：

```
from django.views import generic

from .models import Address

class IndexView(generic.ListView):
    model = Address
    template_name = 'address_list.html'
```

我们只需要从 `generic.ListView` 继承，并创建一个基于类的View，命名为 `IndexView` ，然后为这个类设置两个成员变量，一个为 `model = Address` ，指定我们的generic view需要显示哪一个模型的数据；再设置 `template_name = 'address_list.html'` ，指定显示的模板。

前面已经谈到：使用 generic view 只是减少了 view 的代码量，但对于模板仍然是必不可少的。因此要创建符合 generic view 要求的模板。主要是模板存放的位置和模板文件的名字。

缺省需要的模板文件名为： `app_label/model_name_list.html` ，在这个模板中可以使用 `object_list` 变量访问模型的列表 。

14 在 address 中创建 templates 子目录

15 创建 address/templates/address/list.html

```
<h1>通讯录</h1>
<hr>
<table border="1">
<tr>
  <th>姓名</th>
  <th>性别</th>
  <th>电话</th>
  <th>手机</th>
  <th>房间</th>
</tr>
{% for person in address_list %}
<tr>
  <td>{{ person.name }}</td>
  <td>{{ person.gender }}</td>
  <td>{{ person.telphone }}</td>
  <td>{{ person.mobile }}</td>
  <td>{{ person.room }}</td>
</tr>
{% endfor %}
</table>
```

16 修改 urls.py

将我们的应用的 urls.py include 进去。

```
from django.conf.urls import include, url
from django.contrib import admin
from . import helloworld, add, list, xls_test, login

urlpatterns = [
    url(r'^admin/', admin.site.urls),
    url(r'^$', helloworld.index),
    url(r'^add/$', add.index),
    url(r'^list/$', list.index),
    url(r'^xls/(?P<filename>\w+)/$', xls_test.output),
    url(r'^login/$', login.login),
    url(r'^logout/$', login.logout),
    url(r'^wiki/', include('wiki.urls')),
    url(r'^address/', include('address.urls')),
]
```

可以看到 `r'^address/'` 没有使用 `$` ，因为它只匹配前部分，后面的留给 address 中的 `urls.py` 来处理。

17 启动 server 看效果

通讯录

姓名	性别	电话	手机	房间
limodou	M	1111	11111111111	1111

Django Step by Step (八)

1 引言

上一讲的确很长，但如果看代码你会发现，代码主要在 model 的调整中， `urls.py` 的工作不多，而连一行 view 的代码都没有写。是不是非常方便呢！

那么让我们来继续完善这个通讯录吧。

现在我想完成的是：

- 增加批量导入和导出功能

为什么要批量导入呢？因为一般情况下，我一定是已经有了一个通讯录文件(象以前我说过的Excel文件)，那么现在需要转到 web 上来，难道要我一条条全部手工录入吗？能不能上传文件，自动插入到数据库中去呢？那么就让我们实现一个文件上传的处理吧。

为了简化，我采用csv格式文本文件(这个文件在svn中有一个例子 `data.csv` ，不然就自行生成好了)。

```
abc,M,11,11,11,
bcd,M,11,11,11,
ass,M,11,11,11,
dfsdf,F,11,11,11,
sfas,F,11,11,11,
...
```

2 修改 address/templates/address/list.html

```
<h1 id="title">通讯录</h1>
<hr>
<form enctype="multipart/form-data" method="POST" action="/address/upload/">
上传通讯录文件： <input type="file" name="file"/><br/>
<input type="submit" value="上传文件"/>
</form>
<table border="1">
<tr>
  <th>姓名</th>
  <th>性别</th>
  <th>电话</th>
  <th>手机</th>
  <th>房间</th>
</tr>
{% for person in address_list %}
<tr>
  <td>{{ person.name }}</td>
  <td>{{ person.gender }}</td>
  <td>{{ person.telphone }}</td>
  <td>{{ person.mobile }}</td>
  <td>{{ person.room }}</td>
</tr>
{% endfor %}
</table>
```

3 修改 address/views.py

```
from django.http import HttpResponseRedirect
from django.shortcuts import render_to_response
from django.views.decorators.csrf import csrf_exempt

@csrf_exempt
def upload(request):
    file_obj = request.FILES.get('file', None)
    if file_obj:
        import csv
        from io import StringIO
        try:
            csvfile = StringIO(file_obj.read().decode())
            reader = csv.reader(csvfile)
        except:
            return render_to_response('address/error.html',
                {'message':'你需要上传一个csv格式的文件！'})
        for row in reader:
            objs = Address.objects.filter(name=row[0])
            if not objs:
                obj = Address(name=row[0], gender=row[1],
                    telphone=row[2], mobile=row[3], room=row[4])
            else:
                obj = objs[0]
                obj.gender = row[1]
                obj.telphone = row[2]
                obj.mobile = row[3]
                obj.room = row[4]
            obj.save()

        return HttpResponseRedirect('/address/')
    else:
        return render_to_response('address/error.html',
            {'message':'你需要上传一个文件！'})
```

这里有一个 `upload()` 方法，它将使用 csv 模块来处理上传的 csv 文件。首先查找姓名是否存在于数据库中，如果不存在则创建新记录。如果存在则进行替换。如果没有指定文件直接上传，则报告一个错误。如果解析 csv 文件出错，则也报告一个错误。

报造错误使用了一个名为 error 的模板，我们马上要创建。

4 创建 address/templates/address/error.html

```
<h2>出错</h2>
<p>{{ message }}</p>
<hr>
<p><a href="/address/">返回</a></p>
```

很简单。

5 修改 address/urls.py

```
from django.conf.urls import url

from . import views

urlpatterns = [
    url(r'^$', views.IndexView.as_view(), name='index'),
    url(r'^upload/$', views.upload),
]
```

增加一个 upload 的 url 映射。

6 启动 server 测试

这样导入功能就做完了。那导出呢？很简单了，参考 csv 的例子去做就可以了。不过，并不全是这样，仍然有要修改的地方，比如 csv.html 模板，它因为写死了处理几个元素，因此需要改成一个循环处理。

7 修改 address/templates/address/csv.html

```
{% for row in data %}{% for i in row %}"{{ i|addslashes }}",{% endfor %}
{% endfor %}
```

将原来固定个数的输出改为循环处理。

8 修改 address/templates/address/list.html

增加一个生成导出的 csv 文件的链接

```
<h1 id="title">通讯录</h1>
<hr>
<form enctype="multipart/form-data" method="POST" action="/address/upload/">
上传通讯录文件： <input type="file" name="file"/><br/>
<input type="submit" value="上传文件"/>
</form>
<hr>
<p><a href="/address/output/">导出为csv格式文件</a></p>
<table border="1">
<tr>
  <th>姓名</th>
  <th>性别</th>
  <th>电话</th>
  <th>手机</th>
  <th>房间</th>
</tr>
{% for person in object_list %}
<tr>
  <td>{{ person.name }}</td>
  <td>{{ person.gender }}</td>
  <td>{{ person.telphone }}</td>
  <td>{{ person.mobile }}</td>
  <td>{{ person.room }}</td>
</tr>
{% endfor %}
</table>
```

9 修改 apps/address/views.py

```
from django.http import HttpResponse
from django.template import loader, Context

def output(request):
    response = HttpResponse(content_type='text/csv')
    response['Content-Disposition'] = 'attachment; filename=%s' % 'address.csv'
    t = loader.get_template('address/csv.html')
    objs = Address.objects.all()
    d = []
    for o in objs:
        d.append((o.name, o.gender, o.telphone, o.mobile, o.room))
    c = {'data': d,}
    response.write(t.render(c))
    return response
```

在开始处增加了对 `HttpResponse` , `loader` , `Context` 的导入。然后增加了用于输出处理的 `output()` 方法。

10 修改 address/urls.py

```
from django.conf.urls import url

from . import views

urlpatterns = [
    url(r'^$', views.IndexView.as_view(), name='index'),
    url(r'^upload/$', views.upload),
    url(r'^output/$', views.output),
]
```

增加了对 output 方法的 url 映射。

11 启动 server 测试

Django Step by Step (九)

1 引言

不知道大家有没有对这个通讯录感到厌烦了，希望没有，因为还有一些东西没有讲完呢。

最让我感觉不满意的就是通讯录的显示了，的确很难看，希望可以美化一下。那么主要从这几方面：

- 对姓名进行排序
- 生成分页结果
- 增加css和一些图片

2 修改 address/models.py 实现排序

可以在 model 中增加一个叫 `Meta` 的内类，然后通过对其设置类属性可以用来控制 model 的模型属性。如我们想实现表的排序，可以在 `Meta` 中增加一个 `ordering = ['name']` 的属性即可。它表示按 `name` 进行排序。它可以有多个字段。如果在字段前加'-'表示倒序。修改完毕在浏览器中看一下效果就知道了。models.py的代码如下：

```
from django.db import models

# Create your models here.

class Address(models.Model):
    name = models.CharField('姓名', max_length=20, unique=True)
    gender = models.CharField('性别', choices=(('M', '男'), ('F', '女')),
        max_length=1)
    telphone = models.CharField('电话', max_length=20)
    mobile = models.CharField('手机', max_length=11)
    room = models.CharField('房间', max_length=10, default='')

    def __str__(self):
        return self.name

    class Meta:
        ordering = ["name"]
```

3 修改 templates/address/address/list.html 实现分页显示

```
<html>
<head>
</head>
<body>
  <h1 id="title">通讯录</h1>
  <hr>
  <div>
    {% if is_paginated %}
    <table border="0" width="500">
      <tr align="right">
        <td>
          {% if page_obj.has_previous %}
          <a href="/address?page={{ page_obj.previous_page_number }}">上一页</a>
          {% endif %}
          {% if page_obj.has_next %}
          <a href="/address?page={{ page_obj.next_page_number }}">下一页</a>
          {% endif %}
        </td>
      </tr>
    </table>
    {% endif %}
    <table border="1" width="500">
      <tr>
        <th>姓名</th>
        <th>性别</th>
        <th>电话</th>
        <th>手机</th>
        <th>房间</th>
      </tr>
      {% for person in address_list %}
      <tr>
        <td>{{ person.name }}</td>
        <td>{{ person.gender }}</td>
        <td>{{ person.telphone }}</td>
        <td>{{ person.mobile }}</td>
        <td>{{ person.room }}</td>
      </tr>
      {% endfor %}
    </table>
  </div>
  <table border="0" width="500">
    <tr>
      <td>
        <form enctype="multipart/form-data" method="POST" action="/address/upload/">
          文件导入：
          <input type="file" name="file" />
          <br/>
          <input type="submit" value="上传文件" />
        </form>
      </td>
      <td>
        <p>
```

```
            <a href="/address/output/">导出为csv文件</a>
          </p>
        </td>
      </tr>
    </table>
  </body>
  </html>
```

这时我仍然使用的是 generic view 来处理。但对布局作了简单的调整，将导入和导出的内容移到下面去了。同时增加了对分页的支持:

```
{% if page_obj.has_previous %}
<a href="/address?page={{ page_obj.previous_page_number }}">上一页</a>
{% endif %}
{% if page_obj.has_next %}
<a href="/address?page={{ page_obj.next_page_number }}">下一页</a>
{% endif %}
```

在使用 generic view的 `object_list` 时，它会根据 URL Dispatch 中是否设置了 `paginate_by` 这个参数来决定是否使用分页机制。一会我们会看到在 `urls.py` 的这个参数。一旦设置了这个参数，则 `object_list` 会使用 Django 提供的一个分页处理器来实现分页。它会自动产生分页所用到的许多的变量，这里我们使用了 `has_previous`，`previous`，`has_next`，`next` 这四个变量，还有其它一些变量可以使用。具体的参见 Generic views 文档。

这里是根据是否有前一页和下一页来分别生成相应的链接。对于分页的链接，需要在url中增加一个 Query 关键字 `page` 。因此我的模板中会使用 `page={{ previous }}` 和 `page={{ next }}` 分别指向前一页和下一页的页码。

4 修改address/views.py

```
class IndexView(generic.ListView):
    model = Address
    template_name = 'address/list.html'
    paginate_by = 2
```

我们为 `IndexView` 类增加一个成员变量：`paginate_by` ，指定每一页显示2条记录。

5 启动 server 测试

显示效果为



姓名	性别	电话	手机	房间
abc	M	11	11	11
aksdhf	F	11	11	11
alfj	M	11	11	11
aosfdu	M	11	11	11
ass	M	11	11	11
auf	F	11	11	11
bcd	M	11	11	11
dfsdf	F	11	11	11
ewrqa	F	11	11	11
fafasf	F	11	11	11

下面让我们为它添加一些CSS和图片，让它变得好看一些。

首先要说明一下，我们一直处于开发和测试阶段，因此我们一直使用的都是 Django 自带的 server(其实我个人感觉这个 server 的速度也挺快的)，但最终我们的目的是把它部署到 Apache 上去。现在我们打算增加 CSS 和添加一些图片， Django 提供了这个能力，这就是对静态文件的支持，但是它只是建议在开发过程中使用。真正到了实际环境下，还是让专门的 web server 如 Apache 来做这些事情。只要改一下链接设置就好了。更详细的说明要参见 Managing static files 的文档。同时在 Django 中为了不让你依赖这个功能，特别在文档的开始有强烈的声明：使用这个方法是低效和不安全的。同时当 `DEBUG` 设置(在 `settings.py` 中有这个选项， `True` 表示处于调试期，会有一些特殊的功能)为 `False` 时，这个功能就自动无效了，除非你修改代码让它生效。

6 修改 urls.py

```
from django.conf import settings
from django.conf.urls.static import static

urlpatterns = [
    # ... 其他的URL Pattern ...
] + static(settings.STATIC_URL, document_root=settings.STATIC_ROOT)
```

我们使用static函数，它需要两个参数。第一个参数是通过URL访问静态文件时的相对路径，在 `settings.py` 文件中默认设置为 `STATIC_URL = '/static/'` ，也就是通过 `http://yourhost/static/` 访问静态文件；第二个参数是静态文件在服务器上存放的路

径， `STATIC_ROOT` 就是我将用来存放 CSS 和图片的地方，这里我使用了一个 `STATIC_PATH` ，它从哪里来呢？它是我自已在 `settings.py` 中定义的。在前面有一个导入语句:

```
from django.conf import settings
```

从这里可以看到是如何使用 settings.py 的，我们完全可以自已定义新的东西，并让它在整个项目中生效。

7 修改 settings.py

在最后增加:

```
STATIC_ROOT = os.path.join(BASE_DIR, "collect_static/")

STATICFILES_DIRS = [
    os.path.join(BASE_DIR, "static"),
]
```

STATIC_ROOT这个字段的的目录路径是用来为部署而收集静态文件的地方。 更具体的说呢，当我们执行 `python manage.py collectstatic` 命令的时候，系统会帮我们把所有的静态文件都收集到该目录下。 STATICFILES_DIRS默认是一个空列表，那么这个设置定义了staticfiles app将会遍历的一个附加的位置信息。该值应该设置为一个字符串的列表形式，每个元素都是附加文件目录的绝对路径。

> 注意：这些路径都应该使用unix风格的斜杠，即便是在windows平台上("C:/Users/user/mysite/extra_static_content")

那么我需要在 `newtest` 目录下创建一个 `static` 和 `collect_static` 的目录。

8 创建 newtest/static 目录

这样根据上面 `urls.py` 的设置，我们以后将通过 `/static/XXX` 来使用某些静态文件。

为了美化，我想需要一个 CSS 文件来定义一些样式，同时我还想提供一个 Django Powered 的图片。 在这里有官方提供的图标 。 于是我下了一个放在了 `static` 目录下。同时 CSS 怎么办，自已重头写，太麻烦，反正只是一个测试。于是我下载了 Django 站点用的 css 叫 `base.css` 也放在了 `static` 下面。下面就是对模板的改造。

> 在 SVN 中我放了一个 css 和 gif 图片大家可以使用，不然可能看不出效果。

为了通用化，我新增了一个 `base.html` 它是一个框架，而以前的 `address/list.html` 是它的一个子模板。这样我们就可以了解如何使用模板间的嵌套了。

9 创建 newtest/templates/base.html

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
    <head>
        <meta http-equiv="Content-type" content="text/html; charset=utf-8" />
        <meta http-equiv="Content-Language" content="en-us" />

        <title>Address</title>

        <meta name="ROBOTS" content="ALL" />
        <meta http-equiv="imagetoolbar" content="no" />
        <meta name="MSSmartTagsPreventParsing" content="true" />
        <meta name="Copyright" content="This site's design and contents Copyright (c)
2005  Limodou." />

        <meta name="keywords" content="Python, Django, framework, open-source" />
        <meta name="description" content="Django is a high-level Python Web framework
that encourages rapid development and clean, pragmatic design." />

        <link href="/static/base.css" rel="stylesheet" type="text/css" media="screen"
/>

    </head>

    <body>
        <div id="container">
            {% block content %}content{% endblock %}
        </div>
        <div id="footer">
            <div>
            <img src="/static/djangopowered.gif"/>
            </div>
            <p>&copy; 2005 Limodou. Django is a registered trademark of Lawrence Journ
al-World.</p>
        </div>
    </body>
</html>
```

有些代码也是从 Django 的网页中拷贝来的。特别要注意的是:

```
{% block content %}content{% endblock %}
```

这样就是定了一个可以扩展的模块变量块，我们将在 `address/list.html` 中扩展它。同时对 CSS 和 Django-Powered 的图片引用的代码是:

```
<link href="/static/base.css" rel="stylesheet" type="text/css" media="screen" />
<img src="/static/djangopowered.gif"/>
```

前面都是从 `static` 开始的。这样就将使用我们前面在 `urls.py` 中的设置了。

10 修改 templates/address/address/list.html

```
{% extends "base.html" %}
{% block content %}
<style type="text/css">
  h1#title {
    color: black;
  }
</style>
<div id="header">
  <h1 id="title">通讯录</h1>
</div>
<hr>
<div>
  {% if is_paginated %}
  <table border="0" width="500">
    <tr align="right">
      <td>{% if page_obj.has_previous %}
        <a href="/address?page={{ page_obj.previous_page_number }}">上一页</a>
        {% endif %} {% if page_obj.has_next %}
        <a href="/address?page={{ page_obj.next_page_number }}">下一页</a>
        {% endif %}</td>
    </tr>
  </table>
  {% endif %}
  <table border="1" width="500">
    <tr>
      <th>姓名</th>
      <th>性别</th>
      <th>电话</th>
      <th>手机</th>
      <th>房间</th>
    </tr>
    {% for person in address_list %}
    <tr>
      <td>{{ person.name }}</td>
      <td>{{ person.gender }}</td>
      <td>{{ person.telphone }}</td>
      <td>{{ person.mobile }}</td>
      <td>{{ person.room }}</td>
    </tr>
    {% endfor %}
  </table>
</div>
```

```
<table border="0" width="500">
  <tr>
    <td>
      <form enctype="multipart/form-data" method="POST" action="/address/upload/">
        文件导入：
        <input type="file" name="file" />
        <br/>
        <input type="submit" value="上传文件" />
      </form>
    </td>
    <td>
      <p>
        <a href="/address/output/">导出为csv文件</a>
      </p>
    </td>
  </tr>
</table>
{% endblock %}
```

基本上没有太大的变化，主要是增加了一些 `div` 标签，同时最开始使用:

```
{% extends "base" %}
```

表示是对 base 的扩展，然后是相应的块的定义:

```
{% block content %}
...
{% endblock %}
```

注意，所有扩展的东西一定要写在块语句的里面，一旦写到了外面，那样就不起作用了。

Django 的模板可以不止一次的扩展，但这里没有演示。

11 启动 server 测试

现在你看到的页面是不是象我这样？



12 重要的版权问题

版权是一个可能我们大多数人都不重视的问题，但在实际生产中，这是一个必须重视的问题。许多东西象CSS,图片，甚至可能是一种布局，设计都有可能有版权，在使用这些东西的时候一定要注意相关的说明。不要给自已造成麻烦。如果你不清楚，建议你去找清楚的人，或与所有者联系。特别是对于开源，版权更是一个很重要的东西，因为这是保护我们的武器，希望每个人都重视。特别是对于正式发布的东西，一定要将版权问题交待清楚。

Django Step by Step (十)

1 引言

现在我们看一看所展示出来的页面，你满意吗？还有可以改进的地方。比如性别，它显示出来的直接是数据库的值，而不是对应的“男”，“女”，怎么办。还有表格显示也不是很好看。没说的，改！

最初我想使用 CustomManipulator (Manipulator 是 Django 中用来自动生成元素对应的 HTML 代码的对象，你可以定制它)，但使用 Manipulator 的话，你不能再使用 generic view 了，需要自己去实现 generic view 的某些代码，当然可以 copy and paste，但我目前不想那样做。于是我想到可以扩展 Django 的模板，自定义一个 filter 来实现它。(具体扩展的文档参见 The Django template language: For Python programmers ，你不仅可以扩展filter，还可以扩展 Tag ，还可以设置模板变量，还可以进行块处理等复杂的操作，自己去看吧。)

2 创建 address/templatetags 目录

注意，这个目录要在某个应用的下面，同时它应与 models, views.py在同一层目录下。

3 创建 address/templatetags/init.py 文件

文件为空即可。

4 创建自定义模板文件 address/templatetags/change_gender.py

文件名为你想要装入到模板中的名字。如文件起名为 `change_gender.py` ，那么你将可以在模板中使用:

```
{% load change_gender %}
```

来导入。

5 编辑 change_gender.py

```
from django import template

register = template.Library()

@register.filter(name='change_gender')
def change_gender(value):
    if value == 'M':
        return '男'
    else:
        return '女'
```

先是导入 template 模块，然后生成一个 register 的对象，我将用来它注册我所定义的 filter 。我实现的 filter 将命名为 "change_gender" ，它没有参数(一个filter可以接受一个参数，或没有参数)。当 value 为 M 时返回 男 ，当 value 为 F 时返回 女 。然后调用 register 的 filter 来注册它。这里有两种写法，一种是使用 Python 2.4才支持的 decorator (此行注释掉了)，另一种是使用标准的写法。在使用 decorator 时，如果 filter 方法有多个参数的话，需要指明 name 参数，否则可以直接写为:

```
@register.filter
```

它自动将函数名认为是filter的名字。

6 修改 templates/address/list.html

```
{% extends "base.html" %}
{% block content %}
{% load change_gender %}
<style type="text/css">
h1#title {color:white;}
.mytr1 {background:#D9F9D0}
.mytr2 {background:#C1F8BA}
.myth {background:#003333}
.th_text {color:#ffffff}
</style>
<div id="header">
<h1 id="title">通讯录</h1>
</div>
<hr>
<div id="content-main">
    <table border="0" width="500">
    <tr align="right">
      <td>{% if has_previous %}
        <a href="/address?page={{ previous }}">上一页</a>
        {% endif %} {% if has_next %}
        <a href="/address?page={{ next }}">下一页</a>
        {% endif %}</td></tr>
```

```
    </table>
    <table border="0" width="500" cellspacing="2">
    <tr class="myth">
      <th><span class="th_text">姓名</span></th>
      <th><span class="th_text">性别</span></th>
      <th><span class="th_text">电话</span></th>
      <th><span class="th_text">手机</span></th>
      <th><span class="th_text">房间</span></th>
    </tr>
    {% for person in object_list %}
    <tr class="{% cycle 'mytr1' 'mytr2' %}">
      <td>{{ person.name }}</td>
      <td>{{ person.gender|change_gender }}</td>
      <td>{{ person.telphone }}</td>
      <td>{{ person.mobile }}</td>
      <td>{{ person.room }}</td>
    </tr>
    {% endfor %}
    </table>
    <table border="0" width="500">
    <tr>
    <td>
    <form enctype="multipart/form-data" method="POST" action="/address/upload/">
    文件导入：<input type="file" name="file"/><br/>
    <input type="submit" value="上传文件"/>
    </form>
    </td>
    <td><p><a href="/address/output/">导出为csv文件</a></p></td>
    </tr>
    </table>
</div>
{% endblock %}
```

改动了以下几个地方：

1. 增加了 `{% load change_gender %}` 来导入自定义的 filter 。
2. 增加了几个样式，象 `mytr1` , `mytr2` 等。
3. 显示结果的 table 改为:

```
<table border="0" width="500" cellspacing="2">
```

1. 表头改为:

```
<tr class="myth">
  <th><span class="th_text">姓名</span></th>
  <th><span class="th_text">性别</span></th>
  <th><span class="th_text">电话</span></th>
  <th><span class="th_text">手机</span></th>
  <th><span class="th_text">房间</span></th>
</tr>
```

增加了样式处理

1. 数据显示的 tr 标签改为:

```
<tr class="{% cycle 'mytr1' 'mytr2' %}">
```

使用了 cycle Tag 来处理表格行的样式切换。注意： cycle 处理的是字符串。

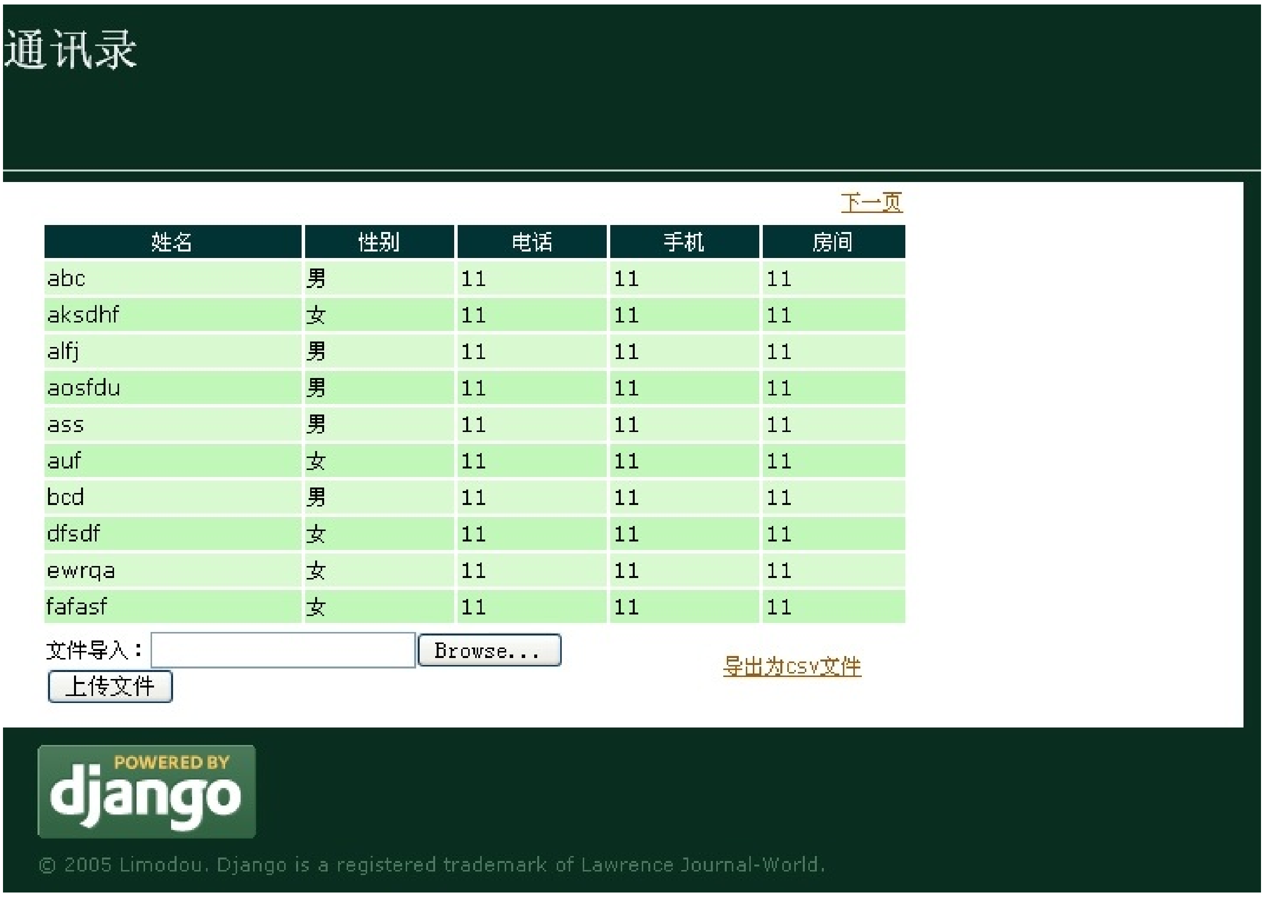
1. 修改 `{{ person.gender }}` 为 `{{ person.gender|change_gender }}`

7 启动 server 进行测试

> 注意，一定要重启。象 templatetags 之类是在导入时处理的，因此如果 server 已经启动再添加的话是不起作用的。其它象增加 app, 修改 settings.py 都是要重启，而修改 urls.py , view, model代码，模板什么的可以不用重启，在必要时 Django 的测试 web server 会自动重启。如果你使用 Apache 的话，估计绝大多数情况下要重启，可能只有修改模板不用吧。不过也仍然可以设置 Apache 以便让每次请求过来时重新装入 Python 模块。

如果一切成功，你会看到 M, F 都改过来了。这里如果你感兴趣还可以改成小图标来表示，点缀一下。

效果画面为：



姓名	性别	电话	手机	房间
abc	男	11	11	11
aksdhf	女	11	11	11
alfj	男	11	11	11
aosfdu	男	11	11	11
ass	男	11	11	11
auf	女	11	11	11
bcd	男	11	11	11
dfsdf	女	11	11	11
ewrqa	女	11	11	11
fafasf	女	11	11	11

Django Step by Step (十一)

1 引言

让我们再仔细看一下这个通讯录，我们知道，如果想增加新的记录，一种方法是通过 admin 界面，这个已经由 Django 自动为我们做好了。我们还可以批量导入，这个是我们实现的。但是这里有风险，为什么？如果什么人都可以导入这可是件不好的事，那么怎么办： 加权限控制 。

Django 自带了一个权限控制系统，那么我们就用它。因此先让我们简单地了解一下 Django 中的权限。同时我希望只有特殊权限的人才可以做这件事情，在前几讲中我们一直使用超级用户，但这并不是个好的习惯。因此让我们先创建个个人用户吧。

2 添加一个个人用户

使用 admin 用户进入管理界面 http://localhost:8000/admin

在 Auth 下有用户一项，点击添加按钮进入添加界面，还挺复杂的。在这里提示是黑体的字段是必输项，其实只有两项是需要我们输的：用户名和密码。用户名好办，密码有一些复杂度的要求：



- 你的密码不能与其他个人信息太相似。

- 你的密码必须包含至少 8 个字符。
- 你的密码不能是大家都爱用的常见密码。
- 你的密码不能全部为数字。

知道了密码的要求之后，那么我们只要填入用户名，密码就行了。



> 注意，职员状态检查框如果不打勾，则你的用户也无法使用，因为他不能登录。也许你担心，如果打勾了，那不是他就能做好多事了吗？其实不然。在 Django 中，创建一个 app 之后都有一些基本的权限会自动生成，而这些除了超级用户，它们是不会自动赋给某个用户的。因此如果管理员不给某个用户关于 app 的使用权限，那么这个用户根本没有办法操纵这些 app ，甚至连看都看不到(大家自已试一下就知道了)。这样他能够做的只是登录，但这也许就够了，有时我们需要的就是一个用户的合法身份，而不是一定要他能做些什么。

`request` 对象提供一个 `user` 对象，你可以根据它来判断当前用户的身份，所属的组，所拥有的权限。我们可以在 view 代码中进行用户身份的检查。

现在我的想法是：限制特殊用户来做这件事。首先我可以在 `settings.py` 中设定这个用户名，然后在 view 中检查当前用户是否是 `settings.py` 中设定的用户。

3 修改 newtest/settings.py

在最后增加:

```
UPLOAD_USER = 'limodou'
```

> 这里请把 limodou 改成你想要的名字。要注意，在后面的测试中你需要按这里指定的名字创建一个用户。

4 修改 address/views.py

```
#...
from django.conf import settings

@csrf_exempt
def upload(request):
    if request.user.username != settings.UPLOAD_USER:
        return render_to_response('address/error.html',
            {'message':'你需要使用 %s 来登录！' % settings.UPLOAD_USER})
#...
```

我们从 django.conf 导出了 settings ，然后在 `upload()` 中判断当前用户名是否是等于 `settings.UPLOAD_USER` 这个用户名，如果不是则提示出错信息。否则继续处理。

好象一切都挺简单，但这里还有一个大问题：能不能自动导向一个用户注册的页面去呢？上面的处理是需要用户进入 admin 管理界面进行注册后，再进行操作。如果没有注册就上传文件，则只会报错。这里我希望实现：如果用户没有注册过，自动显示一个注册页面。如何做呢？

文档中提出了一个方法:

```
from django.contrib.auth.decorators import login_required

@login_required
def my_view(request):
    ...
```

这个方法我试过了，但失败了。主要的原因是：如果你还没有注册，它会自动导向 /accounts/login/ ，而这个URL目前是不存在的。在我分析了 login.py 代码之后，我认为它只是一个框架，并不存在 Django 已经提供好的模板可以直接使用，如果要使用它是不是需要我自己去建一个可以用的模板？没办法，我分析了 admin 的代码之后，最终找到了一种替代的方法:

```
from django.contrib.admin.views.decorators import staff_member_required

@staff_member_required
def upload(request):
```

admin 已经提供了这样的一个方法： `staff_member_required` 。它允许我使用 admin 的登录画面。

一旦把上面的代码补充完整，代码是这样的:

```
from .models import Address

from django.http import HttpResponseRedirect
from django.shortcuts import render_to_response
from django.views.decorators.csrf import csrf_exempt
from django.conf import settings
from django.contrib.admin.views.decorators import staff_member_required

@staff_member_required
@csrf_exempt
def upload(request):
    if request.user.username != settings.UPLOAD_USER:
        return render_to_response('address/error.html',
            {'message':'你需要使用 %s 来登录！' % settings.UPLOAD_USER})
    file_obj = request.FILES.get('file', None)
    if file_obj:
        import csv
        from io import StringIO
        try:
            csvfile = StringIO(file_obj.read().decode())
            reader = csv.reader(csvfile)
        except:
            return render_to_response('address/error.html',
                {'message':'你需要上传一个csv格式的文件！'})
        for row in reader:
            objs = Address.objects.filter(name=row[0])
            if not objs:
                obj = Address(name=row[0], gender=row[1],
                    telphone=row[2], mobile=row[3], room=row[4])
            else:
                obj = objs[0]
                obj.gender = row[1]
                obj.telphone = row[2]
```

```
                obj.mobile = row[3]
                obj.room = row[4]
            obj.save()

        return HttpResponseRedirect('/address/')
    else:
        return render_to_response('address/error.html',
            {'message':'你需要上传一个文件！'})

from django.http import HttpResponse
from django.template import loader, Context

def output(request):
    response = HttpResponse(content_type='text/csv')
    response['Content-Disposition'] = 'attachment; filename=%s' % 'address.csv'
    t = loader.get_template('address/csv.html')
    objs = Address.objects.all()
    d = []
    for o in objs:
        d.append((o.name, o.gender, o.telphone, o.mobile, o.room))
    c = {'data': d,}
    response.write(t.render(c))
    return response
```

基本没有变化，主要是开始的一些地方增加了用户权限的处理。

5 启动 server 测试

在点击上传之后，如果没有注册会进入登录画面。如果已经注册，但用户名不对，则提示一个出错信息。不过，一旦注册出错，没有提供自动重新登录的功能，因此你需要进入 admin 管理地址，然后注销当前用户，再重新上传或先用正确的用户登录。因为是个简单的 app ，没必要做得那么完善。同时还存在的一个问题是，如果你没有注册过，那么点击上传按钮后，将进入登录画面，但如果成功，你上传的文件将失效，需要重新再上传。那么解决这个问题的一个好方法就是：不要直接显示上传的东西，而是先提供一个链接或按钮，认证通过后，再提供上传的页面，这样可能更好一些。

在 User authentication in Django 文档中还有许多的内容，如权限，在模板中如何使用与认证相关的变量，用户消息等内容。

Django Step by Step (十二)

1 引言

如果通讯录中的记录很多，我希望有一种搜索的方法，下面就让我们加一个搜索功能吧。当然，这个搜索功能是很简单的。在 Django 邮件列表中看到 WorldOnline(好象是它)有一个搜索的框架，可以定义哪些模块的哪些字段要参加搜索。这样在处理时会自动将相应的信息加入到搜索数据库中进行预处理。现在这个框架并没有开放源码，而且它底层使用的搜索的东西并不是 Django 本身的。这里我只是对姓名字段进行查找。

2 修改 templates/address/list.html

```
[...]
<hr>
<div id="content-main">
    <table border="0" width="500">
    <tr align="right"><td>
    <form method="GET" action="/address/search/">
    搜索姓名：<input name="search" type="text" value="{{ searchvalue }}"/>
    <input type="submit" value="提交"/>
    </form>
    </td></tr>
    </table>
    <table border="0" width="500">
    <tr align="right">
      <td>{% if has_previous %}
[...]
```

在显示分页的代码上面增加了搜索的处理。

从上面可以看到，条件输入处我增加了一个 `searchvalue` 的变量，希望在提交一个搜索后，显示页面的同时显示当前显示时使用的条件。

由于搜索结果页面也是一个列表页面，我们希望能够用第九讲介绍过的 `generic view` 来显示结果，因为列表页面的处理非常简单：

```
class IndexView(generic.ListView):
    model = Address
    template_name = 'address/list.html'
    paginate_by = 2
```

但是这里存在一个困难：如何把搜索条件，搜索字符串与generic view 相关联呢？通过 `urls.py` 我想是不行的，因为它只从 url 解析，而且对于 QUERY_STRING 是不进行解析的 (QUERY_STRING 是指： `http://example.com/add/?name=test` 中 `?` 后面的东西，也就是 `name=test`)。对于搜索条件，我会使用一个 form 来处理， `method` 会设为 `GET` ，因此生成的 url 中，查询条件正如这个例子，如： `http://localhost:8000/address/search/?search=limodou` 。这样无法变成上面所要用到的参数。

这里我们需要对generic view进行一下扩充，我们需要实现 `get_queryset` 和 `get_context_data` 这两个方法。分别用来指定结果集和模板渲染的参数，我们先来看看新的view方法怎么写：

3 修改 address/views.py

```
class SearchView(generic.ListView):

    template_name = 'address/list.html'
    paginate_by = 2

    def get_queryset(self):
        if self.request.GET.get('search'):
            self.name = self.request.GET['search']
            return Address.objects.filter(name = self.name)
        else:
            self.name = None
            return Address.objects.all()

    def get_context_data(self, **kwargs):
        context = super().get_context_data(**kwargs)
        if self.name:
            context['searchvalue'] = self.name
        return context
```

我们使用 `get_queryset` 方法代替了 `model = Address` ，这样可以更加灵活的定义返回的结果集。

我们使用 `get_context_data` 指定了可以传入到模板中的上下文字典。

`self.request.GET['search']` 从 GET 中得到数据，是一个方便的用法。它将得到提交的查询姓名条件，如果有这个参数，那么我们使用 `filter` 函数对结果进行过滤。如果没有提交，则显示全部数据。

4 修改 address/urls.py

```
urlpatterns = [
    url(r'^admin/', admin.site.urls),
    url(r'^$', helloworld.index),
    url(r'^add/$', add.index),
    url(r'^list/$', list.index),
    url(r'^xls/(?P<filename>\w+)/$', xls_test.output),
    url(r'^login/$', login.login),
    url(r'^logout/$', login.logout),
    url(r'^wiki/', include('wiki.urls')),
    url(r'^address/', include('address.urls')),
] + static(settings.STATIC_URL, document_root=settings.STATIC_ROOT)
```

增加了一个 search 的 url 链接映射。

5 启动 server 测试

感觉这个通讯录也差不多了，现在让我们将其部署到 Apache 上去跑一跑吧。

但部署到 apache 时才知道，问题很多啊。主要问题如下：

- CentOS 7服务器默认自带的Python版本太低

CentOS 7自带的Python版本为Python2.7，我们希望能够使用最新的Python 3.6

- 相对路径的问题

  许多使用相对路径的地方都不对了。必须使用绝对路径。不过这一点对于部署来说的确有些麻烦，好在要改动的地方不多，主要在 settings.py 中。如数据库名字(sqlite3)，模板的位置。

其它的就是要注意的地方了。

6 部署到 Apache 上的体验

只能说是体验了，因为我不是 Apache 的专家，也不是 mod_wsgi 的专家，因此下面的内容只能算是我个人的配置记录，希望对大家有所帮助。

6.1 服务器安装Python 3.6

下面的操作我们都假定环境是CentOS 7的环境，您可以在阿里云、腾讯云等公有云服务商购买ECS服务器，会自动给你安装好相应的操作系统，最后给你一个root的用户名和密码。

使用你自己熟悉的SSH环境，用root用户登录即可，首先安装Python 3.6，执行下面的命令。

```
yum install -y python36
curl https://bootstrap.pypa.io/get-pip.py -o get-pip.py
python36 get-pip.py
pip install virtualenv
```

这样的话你安装的pip会默认使用Python3.6，我们顺手安装好了virtualenv环境。操作系统的Python环境不要安装太多的库文件，都放到自己应用的venv环境中，创建一个虚拟的环境。

6.2 安装 mod_wsgi 模块

mod_wsgi的安装有很多种方法，这里介绍的是官方推荐的办法，使用pip安装，首先需要安装http的开发包，然后使用pip安装mod_wsgi到系统的lib库中，执行下面的命令。

```
yum install -y http-devel python36-devel
pip install mod_wsgi
```

然后我们需要将mod_wsgi安装到apache服务器module中去。

```
cd /etc/httpd/modules
ln -s /usr/lib64/python3.6/site-packages/mod_wsgi/server/mod_wsgi-py36.cpython-36m-x86
_64-linux-gnu.so mod_wsgi.so
```

我们通过在 `/etc/httpd/modules` 下面创建符号链接的方式，让apache在启动的时候自动加载mod_wsgi.so。

然后我们需要在 `/etc/httpd/conf.modules.d` 中创建一个文件，加载mod_wsgi.so，使用 `vi /etc/httpd/conf.modules.d/10-wsgi.conf` 命令创建配置文件，然后录入下面的内容：

```
LoadModule wsgi_module modules/mod_wsgi.so
```

之后使用 `systemctl restart httpd` 重启apache服务即可。

6.2 创建配置文件

假定我们的django工程在/var/www/proc/newtest，那么我们应该创建 `/etc/httpd/conf.d/wsgi.conf`

```
WSGIScriptAlias /newtest /var/www/proc/newtest/newtest/wsgi.py process-group=newtest
WSGIPythonHome /var/www/proc/newtest/venv
WSGIPythonPath /var/www/proc/newtest

<Directory /var/www/proc/newtest/newtest>
    <Files wsgi.py>
        Require all granted
    </Files>
</Directory>

#Deamon模式设置
WSGIDaemonProcess newtest python-home=/var/www/proc/newtest/venv python-path=/var/www/
proc/newtest
WSGIProcessGroup newtest

#静态文件
Alias /newtest/robots.txt /var/www/proc/newtest/static/robots.txt
Alias /newtest/favicon.ico /var/www/proc/newtest/static/favicon.ico

Alias /newtest/media/ /var/www/proc/newtest/media/
Alias /newtest/static/ /var/www/proc/newtest/static/

<Directory /var/www/proc/newtest/static>
Require all granted
</Directory>

<Directory /var/www/proc/newtest/media>
Require all granted
</Directory>
```

`WSGIPythonHome` 是Python运行环境的绝对路径，这里指向我们virtualenv的目录

这里我还设了两个别名，用来指向 `media` 和 `static` 目录。在 `media` 和 `static` 的 `Location` 中设置不进行脚本的解析。

> 上面的 media 路径是指向 Django 在 Python 上的安装目录。你完全可以将其拷贝出来，这样可能要方便得多。另外在 linux 下使用 ln 也相当的方便。

6.3 测试

http://localhost:8888/address

更详细的内容请参见 mod_wsgi 文档。关于 admin 的 media 和 template 好象并不需要配置，大家有什么结果可以告诉我。

同时如果你不想每次重启 Apache 来进行测试，可以将:

```
MaxRequestsPerChild 0
```

改为:

```
MaxRequestsPerChild 1
```

7 后话

上面的步骤是直接把开发的东西发布到了 Apache 中去，但实际中开发与运行可能环境根本不一样，最主要可能就是数据库方面的变化，如果model变化，则有可能要编写数据切换程序。许多实际的问题都需要仔细地考虑。

Django Step by Step (十三)

1 引言

经过一段时间的学习，我想大家对于 Django 的一些基础的东西已经有所了解，但 Django 本身的内容不仅仅如此，它还在发展中，还有许多的专题是我还没有向大家介绍的。因此，随着我和大家一同地学习，我会继续向大家介绍一些更高级的话题。

随着对于web的了解越来越多，我对于 web 上的开发也越来越有兴趣。的确，在实际的工作中我也发现，现在越来越强调团队的管理，许多事情单纯搞一两个人是很困难的，因此如何提高团队工作的一致性和方便性越来越重要，比如：在我所在的项目组，有一些统计信息需要每个人提供，然后进行汇总。目前还是采用手工的方式，这种方式的确简单，但不能自动地进行管理，也不利于以后的归档处理。因此我很希望做成 web 的应用，让每个人可以自由创建项目，提交数据。但就是这样的一个简单的工作，也不是非常简单的事情。如何快速对 Django 加深了解，如何提高开发效率，如何更有效地利用 web 是我更关心的，而不仅仅是做出一个可用的应用来。这包括一系列的 NewEdit 的扩展，及其关知识的积累。

特别让我感兴趣，并且可以极大的提高用户体验的一种 web 技术就是 Ajax 了。它是什么？它是一种技术的总称，包括了 Html, CSS, XML, Javascript 等与 web 相关技术的合集，在我以前的 Blog 也有一些涉及，但那时关注的焦点不在 web 上。现在有机会和时间好好地了解了一下，特别是在 Django 中已经做为实现的目标正在逐步地开展起来，只不过目前还没有可用的东西呈现出来。那么在 Django 的 community 的 blog 上，有人发表了一篇关于使用 dojo (一个 Ajax 的库)来实现在搜索栏中实时输入信息时，可以动态显示与输入信息相匹配的blog 列表的一个例子。他利用 dojo 实现了一个自定义的 widget ，但我感到这种技术对于我这种对于dojo框架不熟悉的人非常有困难。从 blog 上看，实现的过程还是有些复杂。我喜欢先从简单的东西入手。 MochiKit 在 Django 的 Ajax 的讨论中是另一个为大家关注的东西，最大的好处是它的文档最齐全，而且从本人的理解来说，它更简单。而 dojo 则更是提供了很多的 web UI 的控件， MochiKit 基本上没有。不过，在目前情况下我也只是希望体验一下 Ajax 技术，并且做一些简单的应用，而在简单的情况下，我认为 MochiKit 做为入门，作为简单的应用也足够了。

下面就让我以 MochiKit 为基础来向大家介绍一下如何在 Django 中使用它，使用一些简单的 Ajax 技术。

首先让我们关心一下 Ajax 与 Django 的关系。其实 Ajax 本身包含许多的内容，它有浏览器端的显示技术，有与后台通讯的处理，因此与 Django 有关系的其实只有与后台交互那块东西。这样，更多的关于前端显示的技术，如：显示特效，这些都属于 CSS, Javascript的内容，而这些与 Python 本身的关系也不大，因此你还需要掌握这些东西才可以做得更好。也许有机会会有专题和学习和介绍这些方面的东西。

下面的试验主要关注的是前端与后端的交互，也就是如何实现浏览器与 Django 交互，体验不进行页面的刷新(这是Ajax最大的好处，一切都好象在本地进行一样)。

就目前来说， Ajax 与后台交互都是通过浏览器提供的 `XMLHttpRequest` 对象来实现的。这个对象支持同步和异步的调用，但由于 Javascript 本身没有多线程这个东西，因此为了不阻塞浏览器，一般都采用异步方式来调用，而这也是一般的 Ajax 框架提供了默认方式。就目前来说，交互数据也有多种格式，比如：XML, Json , 纯文本/Html。 XML 不用说了，但一般采用 http 协议的 web server 是无法直接支持，因此需要进行转换。同时在浏览器你要同时进行XML的解析，不是非常方便。 Json 是一种数据表示格式，它非常象 Python 的数据类型。而且它只有数据，没有任何的格式，因此数据传输量非常小。再加上处理起来也很方便，在传输上可以直接转换为文本，然后再转换成不同语言的数据结构即可。对于 Python 是非常方便。再有就是文本/Html方式，一种是自定义格式，通过转化为文本进行处理，另一种就是直接使用 html 标记。前一种需要自行做扩展，后一种则是最方便。下面我们将先使用 html 方式，然后再使用 Json 来进行试验。

我设计了一个非常简单的例子：提供一个输入框，用户输入文本，然后点提交，直接在下面显示后台返回的结果。因为我不是 Javascript , CSS 的专家，可能有不对的地方。

2 创建 Ajax 应用

```
manage.py startapp ajax
```

3 修改 ajax/views.py

```
# Create your views here.
from django.http import HttpResponse

def input(request):
    input = request.REQUEST["input"]
    return HttpResponse('<p>You input is "%s"</p>' % input)
```

从这里可以看出，我需要一个 input 字段，然后返回一个HTML的片段。

4 创建 templates/ajax 目录

5 创建 templates/ajax/ajax.html

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html>
    <head>
        <title>Ajax Test</title>
        <script type="text/javascript" src="/site_media/MochiKit.js"></script>
        <script type="text/javascript" src="/site_media/ajax_test.js"></script>
    </head>
    <body>
        <h1>
            Ajax 演示
        </h1>
        <div>
        <form id="form">
        输入：<input type="text" name="input"/>
        <input id="submit" type="button" value="提交" />
        </form>
        </div>
        <div id="output"></div>
    </body>
</html>
```

这个模板将作为初始页面，它用来处理向后台发起请求。在这里它没有需要特殊处理的模板变量，只需要显示即可。但在这里的确有许多要说明的东西。

这是一个标准的 html 的页面，在 head 标签中，它将引入两个 js 文件： MochiKit.js 和 ajax_test.js 。从 url 上可以看出，我将会把它们放在 site_media 下，这个地址就是 media 目录。 MochiKit.js 你需要从 MochiKit 网站下载(最新版本为 1.2)。 MochiKit 下载后有两种格式，一种是单个文件，另一种是分散的文件。我这里使用的是单个文件。

在 html 文件中有一个 form ，它的 id 是 form ，我将用它来查找 form 对象。它有一个文本输入框，还有一个按钮，但这个按钮并不是 submit 按钮。这里有许多与标准的 form 不一样的地方，没有 action, 没有 method ，而且没有 submit 按钮。为什么要这样，为了简单，而且我发现这是 MochiKit 的开发方式。以前写 HTML，CSS, Javascript 和事件之类的处理，我们一般可能会写在一起，但这样的确很乱。在学习了一段 MochiKit 之后，我发现它的代码分离做得非常棒，而这也是目前可能流行的做法。它会在独立的 Javascript 中编写代码，在装载页面时动态地查找相应的元素，然后设置元素的一些属性，如 style ，事件代码等。而在 Html 文档中，你看到的元素中一般就只有 id , class 等内容。这样的好处可以使得处理为以后重用及优化带来方便，同时可以通过编程的方式实现批量的处理，而且也使得 Html 页面更简单和清晰。因为我要使用 Ajax 去动态提交信息，不需要真正的 form 的提交机制，我只是需要用到 form 元素中的数据而已，因此象 action, method 等内容都没有用。 id 是必须的，我需要根据它找到我想要处理的元素对象。

> 不过分离的作法是你的文件将增多，也可能不如放在一个文件中便于部署吧。这是一个仁者见仁，智者见智的作法。

`<div id="output"></div>` 它是用来显示结果的层。

整个处理过程就是：

在装载 html 页面时，会对按钮进行初始化处理，即增加一个 `onclick` 的事件处理，它将完成 Ajax 的请求及结果返回后的处理。然后用户在页面显示出来后，可以输入文本，点击按钮后，将调用 `onclick` 方法，然后提交信息到 Django ，由 Django 返回信息，再由 Ajax 的 deferred 对象(后面会介绍)调用显示处理。

6 创建 media/ajax_test.js

```
function submit(){
    var form = $("form");
    var d = doSimpleXMLHttpRequest('/ajax/input/', form);
    d.addCallbacks(onSuccess, onFail);
}
onSuccess = function (data){
    var output = $("output");
    output.innerHTML = data.responseText;
    showElement(output);
}
onFail = function (data){
    alert(data);
}
function init() {
    var btn = $("submit");
    btn.onclick = submit;
    var output = $("output");
    hideElement(output);
}

addLoadEvent(init);
```

这里有许多是 MochiKit 的方法。

首先让我们看 `addLoadEvent(init);` 它表示将 `init()` 函数加到 `onload` 的响应事件对列中。浏览器在装载完一个页面后，会自动调用 `onload` 事件处理。因此在这里是进行初始化的最好的地方。

`init()` 方法一方面完成对 id 名为 `submit` 的按钮 `onclick` 处理函数的绑定工作，另一个是将 id 为 `output` 的元素隐藏。其实不隐藏也无所谓，因为它本来就是空的，因此你也看不到东西。不过如果有其它的东西这样的处理却也不错。

`$()` 是 MochiKit 提供的一个 `getElement()` 函数别名，它将根据元素的 id 来得到某个对象。

`hideElement()` 是隐藏某个元素。想要显示某个元素可以使用 `showElement()` 。

最重要的工作都在 `submit()` 这个函数中。它首先得到 id 为 `form` 的对象，然后调用 MochiKit 提供的 `doSimpleXMLHttpRequest()` 函数提交一个 Ajax 请求到后台。第一个参数是请求的 url ，第二个如果有的话，应该是 Query String ，即一个 url 的 ? 后面的东西。这里我只是将 form 传给它， `doSimpleXMLHttpRequest()` 会自动调用 `queryString()` (也是 MochiKit 的一个方法)来取得 `form` 中的字段信息。比如你输入了 `aaa` ，那么最终在 Django 你会看到的是:

```
/ajax/input/?input=aaa
```

`doSimpleXMLHttpRequest()` 会返回一个 deferred 对象，它是一个延迟执行对象，在执行了 `doSimpleXMLHttpRequest()` 之后，结果可能当时并没有返回回来，因为这是一个异步调用。因此为了在结果回来之后做后续的处理，我还需要挂接两个异步函数，一个用来处理成功的情况，一个是用来处理失败的情况。 `d.addCallbacks(onSuccess, onFail);` 就是做这件事的。

`onSuccess()` 在 deferred 正确返回后会被调用。 `data` 是 `XMLHttpRequest` 对象本身，它有一个 `responseText` 属性可以使用。这里因为 Django 返回的是 Html 片段，因此我只是简单地将 `output` 对象(用于显示的 div 层)的内容进行了设置。然后调用 `showElement()` 来将层显示出来。

`onFail()` 则只是调用 `alert()` 显示出错而已。

这里有许多 Javascript 和 MochiKit 的东西，如果大家不了解则需要补补课了。其中 MochiKit 的内容在它自带的例子和文档中可以查阅，特别是 MochiKit 自带了一个象 Python shell 一样的命令行解释环境可以进行测试，非常的方便。具体的看 MochiKit 网站上的 ScreenCast 可以了解。

7 修改 urls.py

增加两行:

```
(r'^ajax/$', 'django.views.generic.simple.direct_to_template',
    {'template': 'ajax/ajax.html'}),
(r'^ajax/input/$', 'newtest.ajax.views.input'),
```

前一个使用了 generic view 所提供的 `direct_to_template()` 方法可以直接显示一个模板。后一个则指向了 `views.index()` 方法，它用于在前一个页面点击按钮后与后台交互的处理。

8 安装 ajax 应用

修改 `settings.py`

```
INSTALLED_APPS = (
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.sites',
    'newtest.wiki',
    'newtest.address',
    'newtest.ajax',
    'django.contrib.admin',
)
```

9 启动 server 测试

这样你在文本框中输入内容，点击提交后就会立即在文本框的下面看到结果，而页面没有刷新，这就是 Ajax 就直接的做用。

Django Step by Step (十四)

1 引言

Ajax 因为大量地使用了 Javascript ，而调试 Javascript 的确不是件容易的事，在这方面只有不停地测试，还要靠耐心。而且 Ajax 本身可能还有一些安全方面的东西需要考虑，但这些话题需要你自己去学习了。

在试验了简单的 Html 返回片段之后，让我们再体验一下 Json 的应用吧。为了使用 Json ，我下载了 simplejson 模块。我下载的是 1.1 版本。还可以使用 easy_install 来安装。

如何使用 simplejson 在它自带的文档有示例很简单，下面我们就用它来试验 Json 的例子。

我将在上一例的基础之上，增加一个按钮，这个按钮点击后，会发送一个请求(不带 Json 信息)，然后 Django 会返回一个 Json 格式的表格数据，分为头和体两部分。然后前端动态生成一个表格显示在 `output` 层中。

2 修改 ajax/views.py

```
#coding=utf-8
# Create your views here.
from django.http import HttpResponse

def input(request):
    input = request.REQUEST["input"]
    return HttpResponse('<p>You input is "%s"</p>' % input)

def json(request):
    a = {'head':('Name', 'Telphone'), 'body':[(u'张三', '1111'), (u'李四', '2222')]}
    import simplejson
    return HttpResponse(simplejson.dumps(a))
```

> 由于使用了汉字，前面的 coding=utf-8 一定要加上。

`json()` 是新加的方法。 a 是一个字典，它会被封装为 Json 的格式。这里还使用了汉字，但使用了 unicode 的表示。我发现 simplejson 在处理非 ascii 码时会自动转为 unicode ，但不正确，因此我直接使用了unicode。因此我希望浏览器可以根据这个数据生成表格。

3 修改 templates/ajax/ajax.html

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html>
    <head>
        <title>Ajax Test</title>
        <script type="text/javascript" src="/site_media/MochiKit.js"></script>
        <script type="text/javascript" src="/site_media/ajax_test.js"></script>
    </head>
    <body>
        <h1>
            Ajax 演示
        </h1>
        <div>
        <form id="form">
        输入：<input type="text" name="input"/>
        <input id="submit" type="button" value="提交" />
        <input id="json" type="button" value="JSON演示" />
        </form>
        </div>
        <div id="output"></div>
    </body>
</html>
```

这里只是增加了一个按钮，id 是 `json` 。它将用来触发 Ajax 请求。

4 修改 media/ajax_test.js

```
function callJson(){
    var d = loadJSONDoc('/ajax/json/');
    d.addCallbacks(onSuccessJson, onFail);
}
row_display = function (row) {
    return TR(null, map(partial(TD, null), row));
}
onSuccessJson = function (data){
    var output = $("output");
    table = TABLE({border:"1"}, THEAD(null, row_display(data.head)),
        TBODY(null, map(row_display, data.body)));
    replaceChildNodes(output, table);
    showElement(output);
}
function init() {
    var btn = $("submit");
    btn.onclick = submit;
    var output = $("output");
    hideElement(output);
    var btn = $("json");
    btn.onclick = callJson;
}
```

在最后一行 `addLoadEvent(init);` 前加入上面的内容。对于 id 为 `json` 的按钮的事件绑定方式与上一例相同，都是在 `init()` 中进行的。在 `callJson()` 中进行实际的 Json 调用，这次使用了 MochiKit 提供的 `loadJSONDoc()` 函数，它将执行一个 url 请求，同时将返回结果自动转化为 Json 对象。一旦成功，将调用 `onSuccessJson()` 函数。在这里将动态生成一个表格，并显示出来。

表格的显示使用了 MochiKit 的 DOM 中的示例的方法。 `row_display()` 是用来生成一行的。 `TBODY` 中使用map来处理数组数据。在 MochiKit 中有许多象 Python 内置方法的函数，因为它的许多概念就是学的 Python 。 `replaceChildNodes()` 是用来将生成的结果替换掉 `output` 元素的内容。

5 修改 urls.py

```
(r'^ajax/json/$', 'newtest.ajax.views.json'),
```

增加上面一行。这样就增加了一个 Json 的 url 映射。

6 启动 server 进行测试

这里两个演示共用了 output 层作为显示的对象，你可以同时试一试两个例子的效果。

不过这里有一个问题：只有返回时使用了 Json 。的确是，这样是最简单处理的情况。因为 Json 可以包装为字符串，这样不用在底层进行特殊处理。如果请求也是 Json 的，需要设计一种调用规则，同时很有可能要实现 MiddleWare 来支持。在 Django 中的确有人已经做过类似的工作。不过我目前没有研究得那么深，因此只要可以处理返回为 Json 的情况已经足够了。而且 Django 也正在进行 Ajax 的支持工作，不过可能是以 dojo 为基础的，让我们拭目以待吧。

Django Step by Step (十五)

1 引言

在 Ajax 的试验中，你会看到有一些是用英文写的。下面就让我们学习如何将应用改为支持 i18n 处理的吧。在本讲中我会讲述我实现的过程，同时对一些问题进行讨论。 Django 中 i18n 的实现过程:

1.1 在程序和模板中定义翻译字符串

在程序中就是使用 `_()` 将要翻译的字符串包括起来。这里有几种做法，一种是什么都不导入，这样就使用缺省的方式，另一种是导入 Django 提供的翻译函数。特别是 Django 提供了 Lazy 翻译函数，特别可以用在动态语言的切换。在模板中分几种情况：

可以使用 `{% trans %}` 标签。它用来翻译一句话，但不能在它中间使用模板变量。 如果是大段的文本，或要处理模板变量，可以使用 `{% blocktrans %}{% endblocktrans %}` 来处理。 Django 还支持简单的 Javascript 的 i18n 的处理，但有兴趣自己去看吧。

1.2 生成 po 文件

定义好翻译串之后使用 `bin/make-messages.py` 来生成 po 文件。

Django 支持多层次的处理。比如在整个 Django 的源码项目，在某一个工程，在某一个应用。在不同层次去实现 i18n 时，需要在不同的层次的根目录去执行 `make-messages.py` 。那么可以将 `make-messages.py` 拷贝到相应的目录去执行，特别是在你的工程或应用中。在执行 `make-messasges.py` 时，需要你预先创建 `conf/locale` 或 `locale` 目录，而 `make-messasges.py` 是不会自动为你创建的。那么 `conf/locale` 多用在源码中，象 Django 的源码就是放在 `conf/locale` 中的。 但在运行时，对于自己的项目和应用却是从 **`locale`** 中来找的。因此还是建议你创建 `locale` 来存放 po 文件。

第一次执行时:

```
make-messages.py -l zh_CN
```

这时会生成 `locale/zh_CN/LC_MESSAGES/django.po` 和 `django.pot` 两个文件。

然后你就可以开始翻译了。翻译完成之后，首先要执行类目->设置，将缺省的参数修改一下。主要是：项目名称及版本，团队，团队专用电子邮件，字符集(一般为 utf-8)。这些如果不改， poEdit 在保存时会报错。使用 poEdit 的一个好处是，在保存时它会自动将 po 编译成

mo 文件。

以后再更新时:

```
make-messasges.py -a
```

如果已经有多个语言文件，那么执行时会同时更新这些 po 文件。

1.3 配置

Django 有一系列的策略来实现 i18n 的功能。基本上分为静态和动态。

静态是指在 `settings.py` 中设置 `LANGUAGE_CODE` 为你想要的语言。那么这里要注意，中文的语言编码是 `zh-cn` ，但 `locale` 目录下却是 `zh_CN` 。这是为什么：其实一个是 language(zh-cn) ，一个是 locale(zh_CN) ，在 Django 的 `utils.translation.py` 中有专门的方法可以进行转换。因此在 Django 的程序中使用的是 language 的形式，在目录中却是使用 locale 的形式。一旦设为静态，则它表示是全局性质的，在所有其它的策略失效后将使用这种策略。

而动态是指在运行中对于不同的用户，不同的浏览器的支持的语言可以有不同的语言翻译文件被使用。这种方式需要在 `settings.py` 中安装 `django.middleware.locale.LocaleMiddleware` 到 `MIDDLEWARE_CLASSES` 中去。同时如果你想在实现应用中的翻译文件被使用，也要采用这种方式。

在一个请求发送到 Django 之后，如果安装了 `LocaleMiddleware` ，它会采用下面的策略：

- 在当前用户的 session 中查找 `django_language` 键字。
- 如果没有找到则在 cookie 中查找叫 `django_language` 的值。
- 如果没有找到，则查看 `Accept-Language` HTTP 头。这个头是由浏览器发送给服务器的。
- 如果没有找到，则使用全局的 `LANGUAGE_CODE` 设置。

如果你使用 FireFox 可以在 Tools->Options->Advanced->Eidt Languages 设置你所接受的语言，并且将 `zh-cn` 放在最前面。

上面讲述得还是有些粗，建议你好好阅读 i18n 的文档。

> 国际化处理的文档请参阅： Internationalization 文档

下面开始我们的试验。

2 修改 ajax/views.py

```
#coding=utf-8
# Create your views here.
from django.http import HttpResponse

def input(request):
    input = request.REQUEST["input"]
    return HttpResponse(_('<p>You input is "%s"</p>') % input)

def json(request):
    a = {'head':(unicode(_('Name'), 'utf-8'), unicode(_('Telphone'), 'utf-8')),
        'body':[(u'张三', '1111'), (u'李四', '2222')]}
    import simplejson
    return HttpResponse(simplejson.dumps(a))
```

这里对所有英文都使用 `_()` 进行了封装。但对于 Json 方法，这里我使用 `unicode(_('Name'), 'utf-8')` 进行了转换。

3 修改 settings.py

增加 `LocaleMiddleware`

```
MIDDLEWARE_CLASSES = (
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.locale.LocaleMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.doc.XViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
)
```

这里在文档中对于 `LocaleMiddleware` 的顺序有要求，要求排在 `SessionMiddleware` 之后，但在其它的 Middleware 之前。

> 话虽如此，但我感觉目前顺序影响不大，也许只是个人感觉吧。

4 创建 ajax/locale 目录

5 拷贝 make-messasges.py 到 ajax 目录下

6 执行 make-messasges.py

```
cd ajax
make-message.py -l zh_CN
```

7 使用 poEdit 翻译 django.po 文件

按上面说的先更新 pot 文件，然后修改缺省的参数，再保存。

> 如果你没有 poEdit，或不在 Windows 平台下，那么只好自己去想办法了。同时这里 make-message.py 还需要 Windows 下的 xgettext 工具。可以在 http://code.djangoproject.com/wiki/Localization 找到说明。

这里我没有演示模板的处理。因为 Ajax 所用到的模板没有放在 `ajax` 目录下，而是放在 `templates` 目录下。因此，如果想支持 i18n 的话，目录的布置是一个问题。所以不再试验了。

8 启动 server 测试

是不是都是中文了呢？如果不是，看一看是否浏览器没有设置成接受 `zh-cn` 。

Django Step by Step (十六)

1 引言

Django 中的模板系统可以被自由扩展，如自定义 filter, 自定义 Tag 等。其中 filter 用于对变量的处理。而 Tag 则功能强大，几乎可以做任何事情。我认为 Tag 的好处有非常多，比如：

- 可以简单化代码的生成。一个 Tag 相当于一个代码片段，把重复的东西做成 Tag 可以避免许多重复的工作。
- 可以用来组合不同的应用。将一个应用的展示处理成 Tag 的方式，这样就可以在一个模板中组合不同的应用展示 Tag，而且修改模板也相对容易。

如果要自定义 Tag ，那么要了解 Tag 的处理过程。在 Django 中， Tag 的处理分为两步。

1. 编译。即把 Tag 编译为一系列的 `django.template.Node` 结点。
2. 渲染(Render)。即对每个 Node 调用它们的 `render()` 方法，然后将输出结果拼接起来。

因此自定义一个 Tag ，你需要针对这两步处理来做工作。

在 The Django template language: For Python programmers 文档中讲解了一些例子。大家可以看一下。

那么下面，我将实现一个显示日历的自定义 Tag 。

2 下载 HTMLCalendar 模块并安装

不想全部自已做，因此找了一个现成的模块。去 HTMLCalender 的主页下载这个模块。

然后解压到一个目录下，执行安装:

```
python setup.py install
```

3 下载 HTMLTemplate 模块并安装

然后解压到一个目录下，执行安装:

```
python setup.py install
```

因为上面的 HTMLCalender 需要它才可以运行。去 HTMLCalender 主页下载这个模块。

4 创建 my_alendar 应用

```
manage.py startapp my_calendar
```

这里起名为 my_calendar 。因为如果起名为 calendar 会与系统的 calendar 模块重名。

5 创建 my_calendar/templatetags 目录

```
cd my_calendar
md templatetags
```

> 在 Windows 下是 md, 在 Linux 下是 mkdir 。

6 创建 my*calendar/templatetags/_init*.py 文件

空文件即可

7 创建 my_calendar/templatetags/my_calendar.py 文件

```
from django import template

register = template.Library()

class CalendarNode(template.Node):
    def __init__(self):
        pass

    def render(self, context):
        return "Calendar"

def do_calendar(parser, token):
    return CalendarNode()

register.tag('calendar', do_calendar)
```

上面的代码只是一个空架子。不过让我们仔细地解释一下：

- `register` 与自定义 filter 一样，它将用来注册一个 Tag 的名字到系统中去。
- `CalendarNode` 它是 `template.Node` 的一个子类。每个 Tag 都需要从 `Node` 派生。这个类可以只有 `render()` 方法，用来返回处理后的文本。 `__init__()` 可能是有用的，先预留。
- `render()` 方法接受一个 `context` 参数。这个参数就是在执行模板的渲染时由 View 传入的。不过更复杂的例子是你可以修改 `context` ，这样达到注入新变量的目的。不过本例没有演示。
- `do_calendar()` 是一个由模板处理引擎在发现一个 Tag 的名字之后，将进行调用的方法。那么我们的 Tag 可能在模板中写为 `{% calendar %}` 。这个方法将在下面通过注册过程与一个名字相对应，这里我们想使用 `calendar` 。

  它接受两个参数：

  - `parser` 这是模板处理引擎对象，我们没有用到。
  - `token` 表示 Tag 的原始文本。如果在模板中我们定义 Tag 为 `{% calendar 2006 1 %}` , 那么 `token` 就为 `calendar 2006 1` 。因此你需要对它进一步地处理。

  它将返回一个 Node 的实例，在本例中就是 `CalendarNode` 实例。
- `register.tag('calendar', do_calendar)` 用来注册 Tag 名字和对应的处理方法。

尽管我们没有对 calendar 所带的参数进行处理，但它仍然可以显示。要知道我们还没有使用 HTMLCalender 模块呢。

8 创建 templates/my_calendar 目录

9 创建 templates/my_calendar/calendar.html 文件

```
{% load my_calendar %}
{% calendar 2006 1 %}
```

10 修改 usls.py

增加下面的 url 配置:

```
(r'^calendar/$', 'django.views.generic.simple.direct_to_template',
    {'template': 'my_calendar/calendar'}),
```

11 修改 settings.py 安装 my_calendar 应用

```
INSTALLED_APPS = (
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.sites',
    'newtest.wiki',
    'newtest.address',
    'newtest.ajax',
    'newtest.my_calendar',
    'django.contrib.admin',
)
```

增加了 my_calendar 应用。

12 启动 server 测试

页面上应该显示出 Calendar 的文本。我们在模板中定义的参数没有被用到。因为我们没有眞正调用 HTMLCalender 输出，因此上面只是说明框架是可用的。

下面让我们加入参数的处理。

13 修改 my_calendar/templatetags/my_calendar.py

```
from django import template
import HTMLCalendar

register = template.Library()

class CalendarNode(template.Node):
    def __init__(self, year, mon):
        self.year = int(year)
        self.mon = int(mon)

    def render(self, context):
        return HTMLCalendar.MonthCal().render(self.year, self.mon)

def do_calendar(parser, token):
    try:
        tag_name, arg = token.contents.split(None, 1)
    except ValueError:
        #if no args then using current date
        import datetime
        today = datetime.date.today()
        year, mon = today.year, today.mon
    else:
        try:
            year, mon = arg.split(None, 1)
        except ValueError:
            raise template.TemplateSyntaxError, "%r tag requires year and mon argument
s" % tag_name

    return CalendarNode(year, mon)

register.tag('calendar', do_calendar)
```

主要改动如下：

1. 增加了 `import HTMLCalendar` 的导入。
2. 修改了 `CalendarNode` 的 `__init__()` 方法，增加了两个参数。
3. 修改了 `CalendarNode` 的 `render()` 方法。改成输出一个 Calendar 的表格。
4. 修改了 `do_calendar()` 函数，增加了参数的处理。如果没有输入参数则使用当前的年、月值。否则使用指定的年、月参数。如果解析有误，则引发异常。

> 不过在调试的过程中，的确有一些错误。象开始时我命名为 calendar 目录，结果造成与系统的 calendar 模块重名。然后不得已进行了改名。为什么发现要导入 HTMLTemplate 呢？因为在处理时 HTMLCalender 抛出了异常。但成功后我已经把这些调试语句去掉了。而且发现这些错误 Django 报告得有些简单，你可能不清楚倒底是什么错。因此最好的方法：一是在命令行下导入试一下，看一看有没有导入的错误。另外就是使用 try..except 然后使用 traceback 模块打印异常信息。

14 启动 server 测试

你会看到：

January						
S	M	T	W	T	F	S
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31				

也许感到不好看，没关系，可以通过 CSS 进行美化。当然，这样可能还是不让人满意，比如：不是 i18n 方式的，因此看不到中文。不过这已经不是我们的重点了。掌握了自定义 Tag 的方法就可以自行进行改造了。

同时 HTMLCalender 模块本身可以传入一些链接，这样就可以在日历上点击了。这里不再试验了。有兴趣的可以自己做一下。

Django Step by Step (十七)

1 引言

经过前面许多讲之后，我想大家应该对 Django 的基本开发概念和过程已经有所了解。那么是时候讲一些关于设计方面的东西。首先要声明，目前 Django 基本上还没有什么设计的教程，而我也只能写一些个人体会。

那么这篇教程的体会就是：View, Template and Templatetag

2 View, Temaplte 和 Tag 之间的关系

View 在 Django 中是用来处理请求的，一个 url 请求上来后经过 Django 的处理首先找到这个 url pattern 对应的 View 模块的某个方法。因此 View 是处理请求的起点，同时，在 View 中的方法需要返回，因此它还是一个请求的终点。因此象 Template 和 Tag 只不过是处理中的某些环节。 View 可处理的范围远大于 Template 而 Tag 则只能用在 Template 中。因此从使用范围上说：View > Template > Tag。

Template 是用来输出内容的，目前在 Django 中你可以用它输出文本之类的东西。但象图片之类的非文本的东西，则只能通过 View 来实现，再有如果想在输出时加入一些特殊的 HttpHeader 的控制也只能在 View 中实现。当然，在大多数情况下我们只处理动态的文本生成，其它许多东西都是静态的。象图片之类的可以通过链接来引用。

Tag 是在 Template 中被使用的。它的作用很多，如控制模板逻辑，还可以输出内容并做转换等。Tag 可以自定义，因此你可以在 Tag 中做几乎你想做的有关内容输出的任何事，如从数据库中取出内容，然后加工，输出，许多事情。

在 Django 中提供了一种方便的方法，可以直接将 url 与模板相对应起来。但并不是说你不需要 View 的参与，而是这个 View 的功能是预先写好的，它的作用很简单，就是在 View 方法中直接渲染一个模板并输出。因此说，看上去好象是直接对应，但实际上还是有 View 的处理。比如:

```
(r'^ajax/$', 'django.views.generic.simple.direct_to_template',
    {'template': 'ajax/ajax.html'}),
```

这是在讲 Ajax 的一个 url 的配置，其中使用了 django.views.generic.simple.direct_to_template 这个做好的 View 方法。

3 如何设计

从上面的分析我们可以看出， View, Template, Tag 功能不尽相同，但的确有部分功能的重叠，特别是在文本信息的输出。

如何比较好的选择使用什么来输出呢？

可以从几下以方面考虑：

1. 输出内容

   HTML或文本内容，可以考虑使用 View + Template + Tag ，其它的考虑使用 View

2. 输出范围

   如果是多数据源，比如一个首页，可能包含许多不同的内容，如个人信息统计，Blog展示，日历，相关的链接，分类等，这些信息类型不同，如何比较好的处理呢？可以以 View 为主，即数据在 View 中提供，在模板中考虑输出和布局。但有一个问题，重用不方便。因此采用 Tag 可能更适合。因此对于单一或简单数据源可以只采用 View 和 Template 来实现，而对于多数据源可以采用使用 Temaplate 控制布局，Tag 用来输出单数据源信息。

同时对于多数据源的信息还可以考虑使用 Ajax 技术动态的将信息结合在一起。但使用 Ajax 则需要动态与后台交互，将单数据源的信息组织在一起，这样每个来源都是一个 View 的处理。不过这个有些复杂，这里我们不去考虑它。

因此当你设计结构时，首先考虑实现的内容，是文本的，则可以考虑使用 View, Template 和 Tag。

然后再看是否有重用的需要，有的话，将可重用的部分使用 Tag 来实现，而 View 和 Template 作布局和控制。

4 结论

这里我想到一个问题：我一直想使用 Admin 作为我的数据管理的界面。但经过上面的分析，Admin 目前大多数情况下只处理单一数据表，有些包含关系的，比如一对一，多对一，多对多的可以在一个编辑页面中同时处理多个表的记录，但它还是有可能无法满足复杂的多数据源的表现和编辑问题。因此 Admin 应该可以认为是一个缺省的数据库管理界面，而不完全是一个用户管理界面。因此大多数情况下，你仍然需要自定义管理界面，而不能完全依靠 Admin 。除非你的应用简单，同时对于管理界面的要求不高。

解决了这个问题，于是我们不必太留恋 Admin 的功能，我相信会有一些好的解决方案来满足我们的要求，或者就是我们自已来创建这样的项目。